НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Аналіз тональності текстів за допомогою згорткової нейронної мережі CNN»

Виконав:
студент IV курсу, групи КА-64
Зайвелєв Юрій Ігорович _____

Керівник:
доцент, к.ф-м.н. Яковлева А.П _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к.ф-м.н. Ільєнко А.Б. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Зайвелєв Юрій Ігорович

1. Тема роботи «Аналіз тональності текстів за допомогою згорткової нейронної мережі CNN», керівник роботи доцент кафедри ММСА, к.ф-м.н. Яковлева А.П, затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року

3. Вихідні дані до роботи

1. Операційна система MacOs
2. Частота процесора 2.9 ГГц
3. Мова програмування Python 3
4. Середовище розробки – Jupyter Notebook
5. Бібліотеки, що використовувалися: Pandas, Sklearn, Keras, Tensorflow, Matplotli, NumPy

4. Зміст роботи

1. Проаналізувати існуючі математичні моделі
2. Проаналізувати моделі побудови нейронних мереж CNN
3. Розробити програмний продукт для моделювання CNN та порівняння з баєсівським класифікатором
4. Розробити систему оцінки побудованої моделі
5. Виконати економічний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	к.е.н., доц. Шевчук О. А.	21.04.20	28.05.20

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	13.04.20	
2	Аналіз існуючих моделей	22.04.20	
3	Підбір показників для оцінки моделі	28.04.20	
4	Підбір вхідних даних для моделі	5.05.20	
5	Розробка продукту для моделювання та системи оцінювання моделі	12.05.20	
6	Тестування продукту, отримання результатів	15.05.20	
7	Оформлення дипломної роботи	18.05.20	

Студент Зайвелєв Ю.І.

Керівник Яковлева А.П

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота: 94 с., 6 табл., 2 дод., 42 рис. 54 форм., 5 джерел.

ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, РОСПІЗНАВАННЯ ЕМОЦІЙНОГО ЗАБАРВЕННЯ ТЕКСТІВ, ВЕКТОРНЕ ПРЕДСТАВЛЕННЯ, НАЇВНИЙ БАЄСІВСЬКИЙ КЛАСИФІКАТОР.

Метою даної роботи є побудова штучної нейронної мережі CNN для задачі роспізнавання емоційного забарвлення тексту, також порівняти отримані результати з базовим рішенням у цій сфері для визначення опитамальності використання даного методу обробки інформації. Таким базовим рішенням став наївний баєсівський класифікатор, котрий теж потрібно навчити та порівняти параметри що відображають точність роботи зі значеннями що дала CNN.

Актуальність теми: в наш час дуже актуальною є тема обробки та аналізу тексту у тому числі на його емоційне забарвлення. Ця тема є актуальною у бізнесі для аналізу наприклад відгуків від клієнтів чи виявлення потенційно небезпечних осіб за забарвленням їх текстів тощо. Тобі є необхідним виявляти ефективні моделі для їх вдосконалення та прогресу.

Предметом дослідження є згорткова нейронна мережа котру ми будуємо з комбінації певних шарів, а також варіюючи її гіперпараметри, також є наївний баєсівський класифікатор.

Об’єктом досліджень є підготована за певними стандартами вибірка, котра складається з коментарів та постів соціальної мережі твітер, котрі мають різну довжину та емоційне забарвлення.

Метод дослідження: застосована згорткова нейронна мережа, а також наївний баєсівській класифікатор, а також підходи до навчання CNN такі як навчання моделі Word2Vec для embedding шару CNN, алгоритми навчання мережі були виконані на мови програмування Python 3.

Отримані результати: Була побудована та навчена штучна нейрона мережа CNN з використанням певних підходів, наприклад переднавчання embedding шару, також був навчений наївний баєсівьский класифікатор на тренувальній вибірці коментарів та постів твітер. Далі було оцінено роботу кожного з алгоритмів на тестовій вибірці відносно їх складності реалізації, а наступним кроком було порівняно їх між собою та зроблено висновки про раціональність використання CNN.

ABSTRACT

Thesis: 94 pages, 6 tables, 2 add., 42 images., 54 formulas and 5 references.

CONVOLVED NEURAL NETWORK, RECOGNITION OF EMOTIONAL COLORING OF TEXTS, VECTOR REPRESENTATION, NAIVE BAYES'S CLASSIFIER.

The purpose of this work is to build an artificial neural network CNN for the task of recognizing the emotional color of the text, as well as to compare the results with the basic solution in this area to determine the optimality of this method of information processing. Such a basic solution was a naive Bayesian classifier, which also needs to be taught and compare parameters that reflect the accuracy of work with the values given by CNN.

Relevance of the topic: nowadays the topic of text processing and analysis, including its emotional color, is very relevant. This topic is relevant in business for the analysis of, for example, customer feedback or the identification of potentially dangerous people by the color of their texts, and so on. You need to identify effective models for their improvement and progress.

The subject of the research is a convolutional neural network which we build from a combination of certain layers, and also by varying its hyperparameters, there is also a naive Bayesian classifier.

The object of research is a sample prepared according to certain standards, which consists of comments and posts on the social network Twitter, which have different lengths and emotional color.

Research method: applied convolutional neural network, as well as a naive Bayesian classifier, as well as approaches to CNN learning such as learning the Word2Vec model for embedding the CNN layer, network learning algorithms were performed in Python 3 programming languages.

Results: An artificial neural network CNN was built and trained using certain approaches, such as pre-learning embedding layer, and a naive Bayesian classifier

was trained on a training sample of comments and Twitter posts. Next, the work of each of the algorithms on the test sample was evaluated in relation to their complexity of implementation, and the next step was to compare them with each other and draw conclusions about the rationality of using CNN.

ЗМІСТ

ПЕРЕЛІК УМОНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

CNN – Convolutional neural network

MNB – Multimonial naive bayes

Nan – not a number

ReLu – Rectified linear unit

ВСТУП

Зчитуваня, аналіз та всякого роду обробка інформації є не від'ємною складовою сучасного світу, без якої ми не можемо уявити подальший розвиток людства. Кількість інформації що отримує людина, росте кожен день, навіть якщо ми цього не хочемо. На щастя вся ця інформація може бути оброблена та представлена у зручному для людини вигляді різноманітними системами. В даний момент найбільш простим та розвиненим способом є використання нейронних мереж. Використовуючи нейронні мережі аналізують великі обсяги інформації. Обробляють навіть інформацію що представлена не у вигляді великих масивів даних, а як зображеня. У представленіх роботі ми розглянемо використання нейронних мереж для аналізу тональності тексту.

В інтернеті постійно виникає велика кількість контенту різного виду – коментарі, фото, відео та аудіо записи. В часності коментарі котрі лишають користувачі соціальних мереж та форумів піддаються аналізу, адже там підіймаються найрізноманітніші теми – спорт, політика, робота, бізнес, навчання, наука, технології та інші. Все це цікавить сучасні компанії і не тільки у сфері інформаційних технологій, завдяки цьому виникає потреба емоційного аналізу тональності тексту. Для цього використовують методи аналізу тональності тексту. Тож вирішення цієї проблеми є досить розповсюдженим та важливим питанням в наші дні.

Аналіз тональності тексту може допомогти навчити інформаційну систему сприймати мову, а також використовувати її на рівні близькому до людини. Також аналіз може допомогти покращити аналіз тексту, його переклад та вилучення сенсу адже буде враховуватись емоціональний тон тексту.

При виконанні даної роботи потрібно створити бінарний класифікатор, котрий буде вирішувати яким є представлений текст. Для класифікації тексту

було обрано два типи емоційного забарвлення тексту – позитивний та негативний, а для досягнення поставленої цілі використано способи та векторні моделі представлення.

Як результат виконання отримана модель нейронної мережі CNN, котра дає можливість з певною точністю визначити емоційну тональність тексту, а також порівняти з іншими готовими реалізаціями методів у плані їх ефективності.

1 ОСНОВНІ СКЛАДОВІ ТА АЛГОРИТМИ НЕЙРОННИХ МЕРЕЖ

1.1 Основні складові нейронних мереж

Штучні нейронні мережі побудовані по принципу біологічної нейронної мережі, котрі представляють собою мережі нервових клітин, котрі виконують певні фізіологічні функції. Складовою частиною нейронної мережі як очевидно є нейрон (рисунок 1.1).

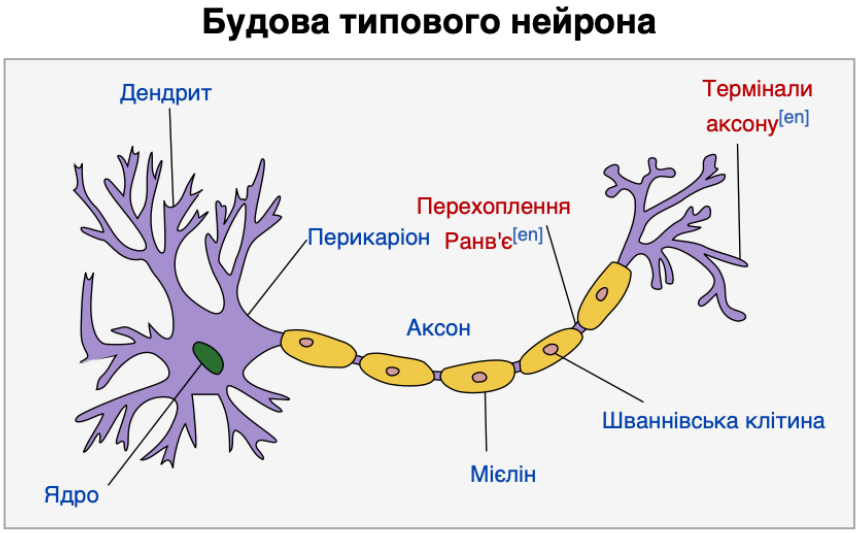


Рисунок 1.1 – будова нейрона

Нейрон має наступні функції

- Синапси приймають інформацію.
- Інтегративна функція.
- Імпульс котрий досяг кінця аксону, провокує медіатор передати збудження наступному нейрону.
- По аксону проходить інформація до синапсу.

Синапс – це зв’язок у мережі по котрому вихідний сигнал з одного нейрону потрапляє на вхід інших нейронів. Зв’язки що мають додатню вагу називають збудливими зв’язками, а ті що мають негативну вагу називають

гальмівними. У нейронної мережі штучний нейрон - це деяка нелінійна функція, аргументом якої є лінійна комбінація всіх вхідних сигналів. Така функція називається активаційною. Потім результат активаційної функції відправляється на вихід нейрона. Об'єднання таких нейронів і називають штучною нейронною мережею. На відміну від справжнього нейрона, штучний складається з нелінійного перетворювача та суматора. Тож бачимо, що штучні нейрони є досить простими та однотипними елементами нейронної мережі, котрі певним чином повторюють роботу нейронів нашого мозку.

Функція активації – це функція залежності вихідного сигналу нейрона від вхідного. Зазвичай для активаційної функції обирають монотонно зростаючу функцію, наприклад гіперболічного тангенсу з проміжку -1;1 чи сигмоїду з проміжку 0;1. Нейрон характеризується своєю функцією активації. Для певних алгоритмів навчання потрібно щоб функція активації буда також диференційовною.

Нижче наведені три основні функції активації:

- Сигмоїдальна активаційна функція (рисунок 1.2,1.3). Ця функція є найпопулярнішою серед активаційних функцій, вона є швидкозростаючою, а також вона тримає певний баланс між лінійною та нелінійною поведінкою. Нижче бачимо вираз котрим можна описати дану функцію, а також її графік:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Рисунок 1.2 – функція сигмоїдальна активаційна функція

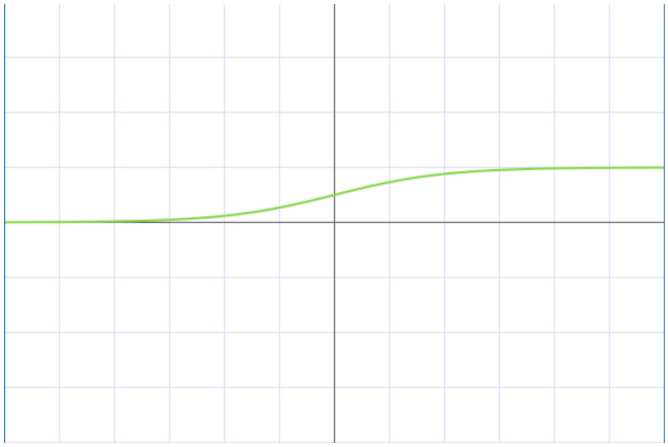


Рисунок 1.3 – графік сигмоїдальної функції

- Гіперболічний тангенс. Нижче бачимо вираз котрим можна представити функцію, а також її графік(рисунок 1.4,1.5):

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

Рисунок 1.4 – функція гіперболіний тангенс

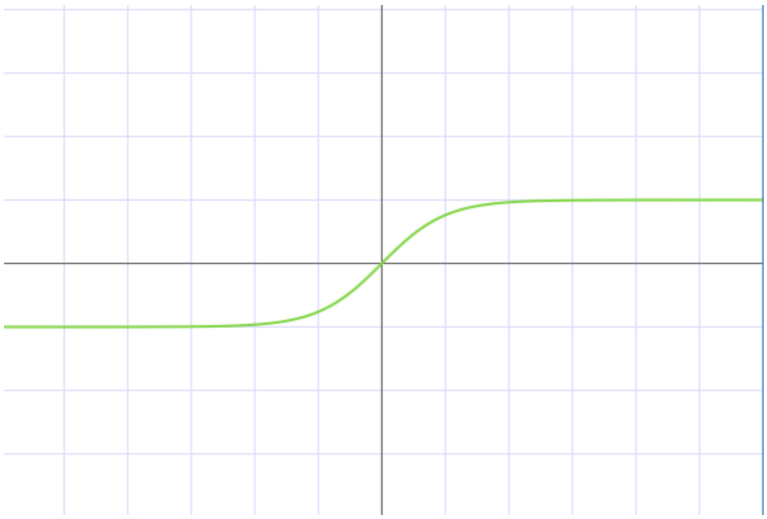


Рисунок 1.5 – графік гіперболічного тангенсу

- Функція одиночний стрибок є кусково-лінійною фукцією. Тож якщо значення на вході менше ніж задане порогове значення, то

функція приймає своє мінімальне значення, у іншому випадку максимальне(рисунок 1.6,1.7).

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & else \end{cases}$$

Рисунок 1.6 – функція одиночний стрибок

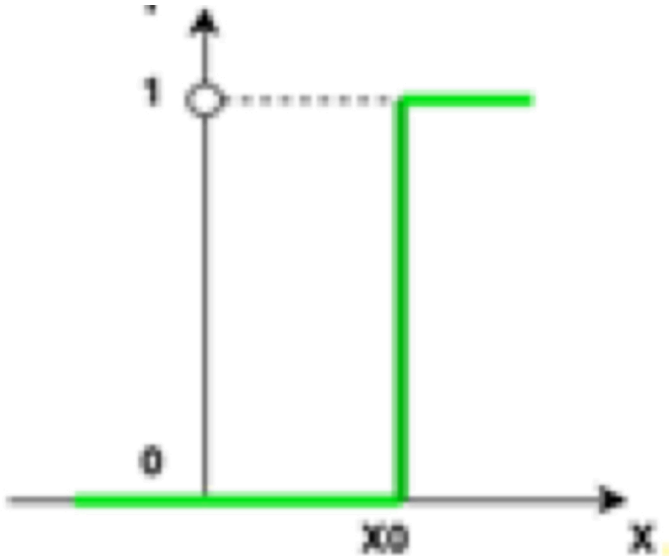


Рисунок 1.7 – графік функції одиночний стрибок

Персептрон - це тип штучного нейрона, що розробив Френк Розенблат в 1950-их і 1960-их роках. Спочатку необхідно маємо розглянути структуру і принцип роботи персептрону. Персептрон приймає на вхід значення $x1$, $x2$ ... і видає бінарний результат. Розенблат запропонував використовувати ваги – це числа, що виражають важливість вкладу кожного входу в кінцевий результат(0 або 1). Зважена сума (або ваги) порівнюється із заданим граничним значенням, котре теж є параметром нейрона, і за результатами порівняння визначається, буде виданий 0 або 1. Нижче можемо бачити схематичне зображення одношарового персептрону(рисунок 1.8).

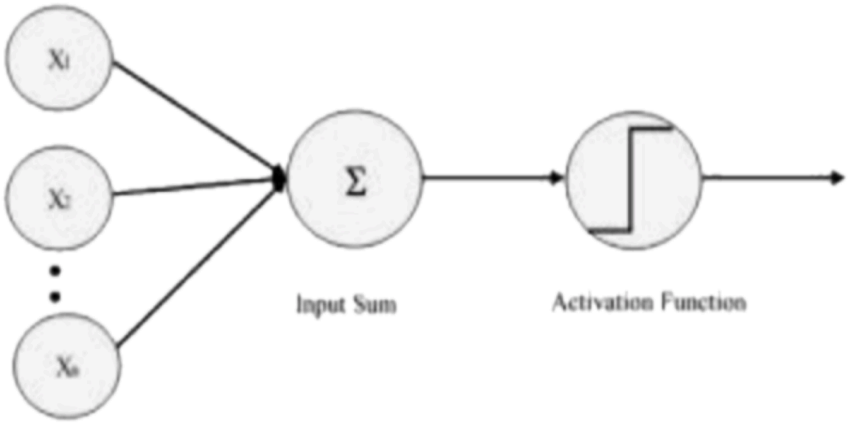


Рисунок 1.8 – одношаровий персептрон

Розенблат запропонував використовувати ваги - числа, що виражають важливість вкладу кожного входу в кінцевий результат. Зважена сума NET (або ваги) порівнюється з граничним значенням (T - threshold), і за результатами визначається, чи буде виданий 0 або 1 (рисунок 1.9).

$$OUT = \begin{cases} 1, NET \geq T \\ 0, NET < T \end{cases},$$

Рисунок 1.9 – приклад використання ваг

Персептрон може бути класифікований наступним чином:

- З граничною активаційною функцією.
- З прямим розповсюдженням сигналу.
- Той що має один прихований шар.

Навчання персептрону полягає в зміні ваг в процесі навчання. Існують наступні види персептронів:

- Одношаровий персептрон має елементи на вході котрі одразу пов'язані з вихідними системою ваг. Він є мережею прямого поширення, у таких мереж немає зворотнього зв’язку бо сигнал

поширюється напряму з вхідного шару до вихідного де ми отримуємо результат опрацювання сигналу.

- Багатошаровий персептрон по Розенблату має додаткові приховані шари.
- Багатошаровий персептрон по Румельхарту має додаткові приховані шари, а навчання проводиться за методом що називається зворотнє поширення помилки.
- Персептрон що має лишень один прихований шар.

Сигмоїдальні нейрони схожі на перцептрони, однак невеликі зміни в їх вагах і зсувах незначно змінюють вихід нейрона.

Цей факт дозволяє мережі з сигмоїдальних нейронів навчатися. На вхід сигмоїдальна нейрона подаються будь-які значення між 0 і 1. На виході також видається значення між 0 і 1, так як в якості активаційної функції використовується сигмоїда, що є нелінійною.

Чим більше β (параметр нахилу сигмоїдальної функції активації), тим сильніше крутизна графіку. При $\beta \to \infty$ сигмоїда прямує до функції Ховісайда.

Важливою властивістю сигмоїдальної функції є її диверенційовність. Застосування неперервної функції активації дозволяє використовувати при навчанні градієнтні методи.

Нейрони можна розділити на групи в залежності від їх положення в мережі:

- вхідні нейрони приймають вихідний вектор даних;
- в проміжних нейронах відбуваються основні обчислювальні операції – процес навчання;
- вихідні нейрони - результат роботи мережі.

Нейрон зміщення або bias нейрон - це вид нейронів, що використовується в більшості нейромереж. Суть цього типу нейронів полягає в тому що вони не мають вхідних синапсів, а також вхід та вихід цього типу завжди дорівнє одиниці. Нейрони зміщення можуть, або бути присутнім в

нейронної мережі по одному на шарі, або повністю відсутні, тож часткової присутності нейронів зміщення у мережі бути не може, наприклад 50 на 50 (червоним кольором на зображенні виділено ті зв’язки та позиції нейронів котрих бути не може) (рисунок 1.10). Тож, їх можна використовувати на вхідному шарі і всіх прихованих шарах, але не на вихідному шарі.

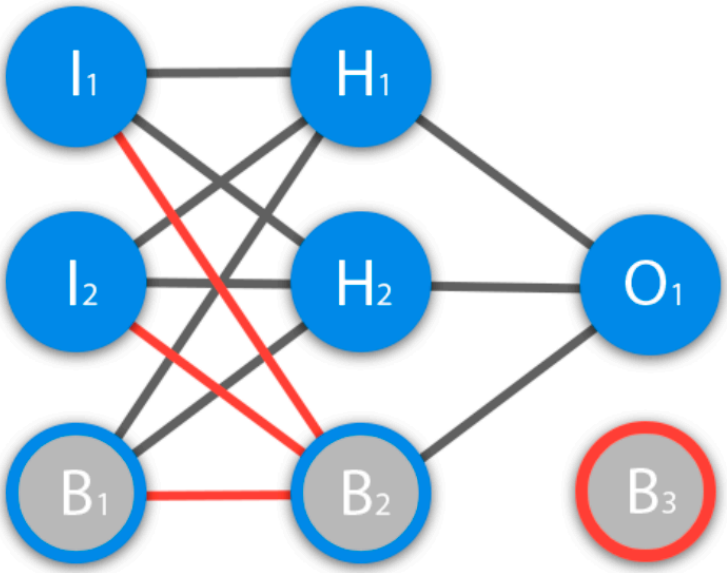


Рисунок 1.10 – схема нейронної мережі

Нейрон зміщення потрібен для того, щоб мати можливість отримувати результат, шляхом зсуву графіку функції активації вправо або вліво по осі X. Також нейрони зміщення допомагають в тому випадку, коли всі вхідні нейрони отримують на вхід 0 і незалежно від того які у них ваги, вони всеж передадуть на наступний шар 0, але не тоді коли в шарі є нейрон зміщення, адже ми додамо начення нейрону зміщення(1) на вагу. Наявність або відсутність нейронів зміщення - це гіперпараметр. По суті, ми самі повинні вирішити, чи потрібно нам використовувати нейрони зміщення чи ні, прогнавши нейронну мережу з нейронами змішання і без них і порівнявши результати. Так як його вихід завжди дорівнює 1, то можна просто уявити що у нас є додатковий синапс з вагою і додати до суми ця вага без згадки самого нейрона.

Іноді на схемах не позначають нейрони зміщення, а просто враховують їх ваги при обчисленні вхідного значення наприклад (рисунок 1.11):

input = H1*w1+H2*w2+b3
b3 = bias*w3

Рисунок 1.11 – приклад використання нейрону зміщення

1.2 Архитектура нейронних мереж

Базовим елементом штучної нейронної мережі є штучний нейрон. Ці елементи пов’язуються зв’язками та утворюють нейронну мережу. Вона в свою чергу дає можливість досить ефективно обробляти інформацію, а також пристосовуватись до змін зовнішнього середовища. Під час роботи штучної нейронної мережі відбувається перетворення вхідного вектору значень(сигналів) на вихідний. Сам процес перетворення визначається характеристикою нейронів, а також структурою, архітектурою та методом тренування нейронної мережі. Основними характеристиками штучної нейронної мережі є:

- Парадигма нейронної мережі – це спосіб використання та навчання нейронної мережі.
- Структура штучної нейромережі – це те як пов’язані нейрони між собою та взаємодіють.
- Архітектура штучної нейромережі – це тип або типи нейронів у мережі та те як вони між собою взаємодіють на пов’язані. Варто зазначити що на основі однієї архітектури можуть бути реалізовані різні парадигми та навпаки.
  Виділяють наступні архітектурні рішення нейронних мереж

- Слабкозв’язні – це нейрони пов’зані тильки з сусідніми (рисунок 1.12).
- Повнозв’язні – це коли кожен нейрон подає сигнал на всі інші нейрони, в тому числі собі (рисунок 1.13).

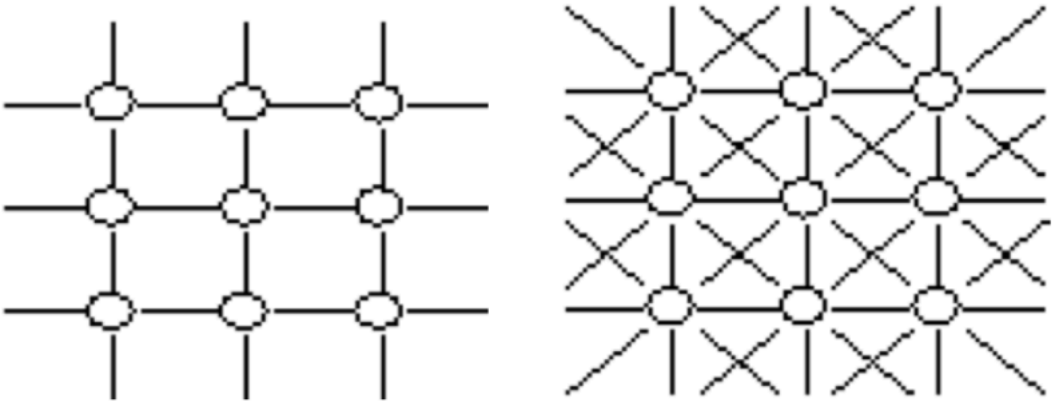


Рисунок 1.12 – Слабозв'язані нейромережі

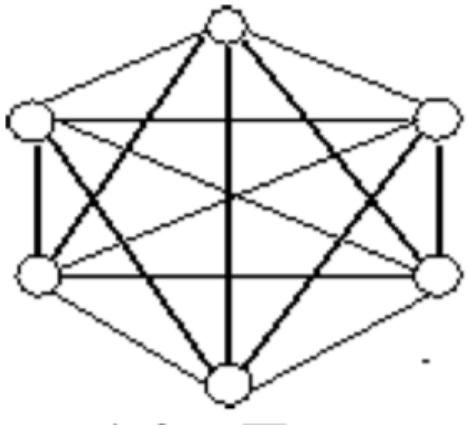


Рисунок 1.13 – Повнозв'язані нейромережі

Розглянемо задачу навчання з учителем. Дано безліч тренувальних прикладів X з мітками (labels) Y. Нейронні мережі визначають нелінійну гіпотезу hW, b (x) з параметрами W і b (рисунок 1.14).

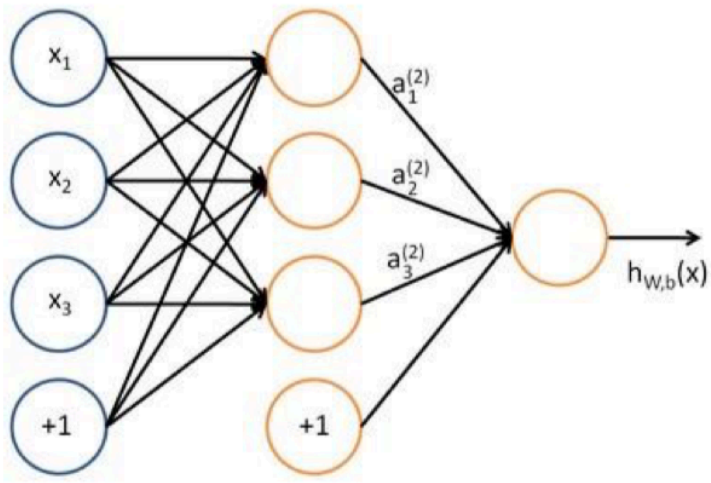


Рисунок 1.14 – приклад мережі

Перший шар на зображенні шар називається вхідним, а останній правий шар - вихідним. Шар посередині є прихованим і називається так через те, що його значення не спостерігаються в тренувальних прикладах. Таким чином в даній мережі ми спостерігаємо елементи входу, приховані елементи і один вихідний елемент. Нехай nl - кількість шарів в мережі, цьому випадку три. Параметри мережі (W, b) = (W (1), b (1), W (2), b (2)). Результат застосування функції активації позначається ai для i-ого елемента. Маємо наступне:

$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_{(1)}^{1}\right) \quad (1.1)$$

$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_{(1)}^{2}\right) \quad (1.2)$$

$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_{(1)}^{3}\right) \quad (1.3)$$

де $a_i^{(j)}$ – значення виходу мереж

W – ваги мереж

b – значення нейрону зміщення

Мережею прямого поширення називаються штучні нейронні мережі, які використовують вихідні данні одного шару як данні для входу наступного шару.

1.3 Навчання нейронної мережі

Розглянемо загальні поняття в навчанні нейронних мереж. Епоха - прямий і зворотний прохід по всім тренувальним прикладам що ми маємо.

Розмір серії (також batch) - кількість тренувальних прикладів для однієї ітерації прямого і зворотного проходів.

Кількість ітерацій - кількість проходів: кожен прохід використовує приклади. Один прохід = прямий прохід додати зворотній прохід.

Тобто маючи тисячі прикладів, batch = 500, нам буде потрібно дві ітерації, щоб завершити одну епоху.

Навчання штучних нейронних мереж це багатопараметрична задача нелінійної оптимізації з точки зору математики.

Алгоритм зворотнього поширення помилки визначає стратегію підбору ваг багатошарової мережі із застосуванням градієнтних методів оптимізації. Оскільки цільова функція, зазвичай визначається як квадратична різниця суми між фактичними і очікуваними вихідними значеннями, є безперервною, градієнтні методи оптимізації є ефективними при навчанні мережі. У процесі навчання багатошарової нейронної мережі ми повинні обчислити вектор градієнта щодо параметрів всіх шарів мережі. Алгоритм у загальному випадку:

- Ініціалізувати ваги випадковими значеннями котрі близькі до нуля.
- Подати на вхід мережі вектор для навчання.
- Отримати вихід мережі.
- Визначити різницю між отриманим виходом та потрібним значенням виходу
- Виправити ваги мережі для покращення результату
- Попередні кроки потрібно повторювати поки не досягнемо прийнятного значення помилки

Квадратична помилка цільової функції (squared-error cost function) для одного вхідного прикладу буде обислюватись наступним чином:

$$J(W, b, x, y) = ½ \|h_{w,b}(x) - y\|^2 \quad (1.4)$$

де h – вихід мережі,

y – значення лейблів.

Якщо розглядати випадок для m прикладів то наша цільова функція буде виглядати наступним чином

$$J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W, b, x_i, y_i)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ij}^l)^2 \quad (1.5)$$

де h – вихід мережі,

y – значення лейблів.

Зрозуміло що перший доданок це сума квадратів помилок прикладів, другий це член регуляції котрий дозволяє зменшити значення ваг і запобігти перенавчання. Параметр регуляризації ваг λ використовують для перевірки відносної значущості частин даного виразу. У завданнях бінарної класифікації у представлений 0 або 1 (так як сигмоїдальна функція видає значення в межах [0; 1], проте при використанні гіперболічного тангенса маркерами класів були б -1 і 1). Завдання - мінімізувати J (W, b). Для навчання нейронної мережі необхідно ініціалізувати кожен параметр W(l) і b(l) малими випадковими величинами(котрі близькі до нуля), а потім застосувати алгоритм оптимізації.

Кожна ітерація градієнтного спуску оновлює параметри таким чином (рисунок 1.15):

$$W_{ij}^{(l)} = W_{ij}^{(l)} - a\frac{\partial}{\partial w_{ij}^{(l)}} J(W, b) \quad (1.6)$$

$$b_i^{(l)} = b_i^{(l)} - a\frac{\partial}{\partial b_i^{(l)}} J(W, b) \quad (1.7)$$

де α – швидкість навчання,

b – зміщення,

J – цільова функція

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J\left(W, b, x^i, y^i\right)\right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J\left(W, b, x^i, y^i\right)$$

Рисунок 1.15 – знаходження похідних

Функція перехресної ентропії використовується як функції втрат: y'-передбачені значення, yi - очікувані значення:

$$L(x, y) = -\frac{1}{n}\sum_{i=1}^{n} y^i \log a\left(x^i\right) + \left(1 - y^i\right) \log 1 - a\left(x^i\right) \quad (1.8)$$

де a – вихід шару,

y – потрібні значення,

n – кількість ваг.

L1-регуляризація: відбувається зміна нерегуляризованій цільової функції шляхом додавання суми абсолютних значень ваг:

$$J = J_0 + \frac{\lambda}{n}\sum_W |W| \quad (1.9)$$

де n – кількість ваг,

λ - параметр регуляції,

W - ваги

При використанні L1-регуляризації відбувається прагнення одного або більше вагових значень до 0.0, тому відповідна функція більше не потрібно. Цей ефект називається селекцією функцій (feature selection);

L2-регуляризація (також відома як weight decay) На відміну від L1-регуляризації, в L2 ваги зменшуються на величину, пропорційну ваг:

$$J = J_0 + \frac{\lambda}{2n}\sum_W W^2 \tag{1.10}$$

де n – кількість ваг

λ - параметр регуляції,

W – ваги

Dropout не впливає на значення цільової функції: змінюється структура мережі. Кожен нейрон видаляється з мережі з певною ймовірністю p. За отриманою прорідженої мережі робиться зворотне поширення помилки, для решти ваг робиться градієнтний крок. Після цього вилучені нейрони відновлюють в мережі. При навчанні нейроної мережі вихід кожного нейрона домножується на (1-p). Так буде отримано матсподівання відповіді мережі по її 2N (де N - кількість нейронів в мережі) архітектурам. Навчена таким чином мережу є результатом усереднення 2N мереж. Окрема нейронна мережа, навчена за допомогою раннього зупинки, має дуже велику помилку, однак усереднення декількох нейронних мереж призводить до істотного зниження помилки;

У випадку з використанням пакетного градієнтного спуску функція втрат(loss) обчислюється для всіх даних разом узятих після закінчення епохи, а далі змінюють ваги нейронів, то при використанні стохастичного методу вагові коефіцієнти змінюються після обчислення виходу останього шару(виходу) на одному з прикладів для навчання.

Недоліком використання пакетного градієнтного спуску є його потрапляння і подальше застрявання в локальних мінімумах. Знаючи те, що стохастичний метод градієнтного спуску працює повільніше пакетного, він здатний вийти з локального мінімуму, що дає змогу краще навчити нейромережу.

Це спосіб знаходження локального мінімуму або максимуму функції за допомогою руху уздовж градієнта. Розглянемо графік на якому по осі x будуть значення ваги нейрона (w) а по осі y – помилка котра відповідає цій вазі (e) (рисунок 1.16).

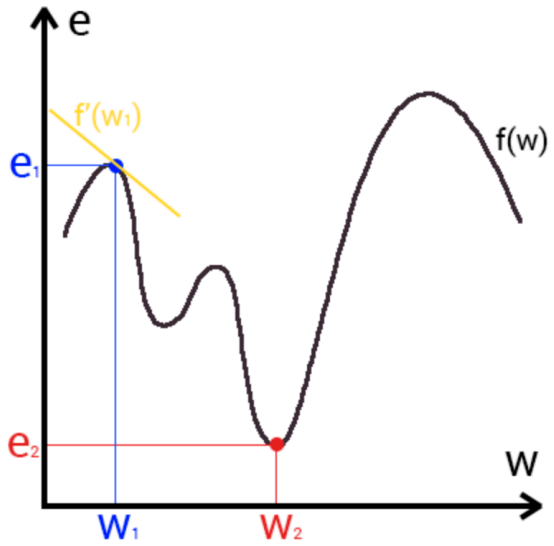


Рисунок 1.16 – графік ваги нейрона та помилки

Бачимо, що функція представлена на графіку є залежністю помилки від вибраної ваги. Точка W2 є глобальним мінімумом, саме тут ми маємо найменшу помилку і як результат ми отримаємо найкращий вихід. Знайти ж цю точку нам допоможе метод градієнтного спуску (жовтим на графіку позначений градієнт). Відповідно у кожного ваги в нейронної мережі буде свій графік і градієнт і у кожного треба знайти глобальний мінімум.

Градієнт - це вектор котрий визначає крутизну схилу і вказує його напрямок щодо будь-якої з точок на поверхні або графіку. Щоб знайти градієнт потрібно взяти похідну від графіка по даній точці (як це і показано

на графіку). Рухаючись у напрямку котрий вказує градієнт та ми будемо плавно спускатися до низини, а саме нашого мінімуму.

Недоліки градієнтного спуску

Основні труднощі навчання штучних нейронних мереж полягає в методах виходу з локальних мінімумів. Недоліками градієнтного спуску при навчанні мережі є:

- Параліч мережі

Значення ваг мережі в результаті корекції можуть стати дуже великими величинами. Оскільки помилка, котра посилається назад в процесі навчання, пропорційна похідній стискає функцію, то може статися так що процес навчання майже зупиниться. Цьому можна запобігти, зменшуючи крок η, однак процес навчання буде відбуватися довше.

- Розмір кроку

Якщо значення кроку не змінювати, і воно досить мале, то метод може сходитися дуже повільно. Якщо крок занадто великий, то у такому випадку може відбутися параліч мережі. Необхідно змінювати значення кроку: збільшувати до того моменту, поки не зупиниться поліпшення оцінки у напрямку антиградієнту або зменшувати, у випадку коли оцінка не поліпшується.

1.4 Глибокі нейронні мережі

Глибокими нейронними мережами є мереж котрі складаються з декількох(більше одного) прихованих шарів. Так як кожен прихований шар проводить обчислення на основі вихідних даних попереднього шару, глибока мережа має набагато більшу репрезентативну потужність (вона може представляти більш складні функції), ніж з одним прихованим шаром. При

навчанні такої нейронної мережі потрібно використовувати нелінійну активаційну функцію в кожному з прихованих шарів.

Головним плюсом глибинних мереж є представлення великої кількості функцій досить зжато. Можна представити, що існують функції, які k-шарова мережа може представляти стисло, а (k-1) шарова мережа не може цього зробити, якщо вона не має експоненційно великої кількості елементів у шарі. Приклад такої мережі (рисунок 1.17).

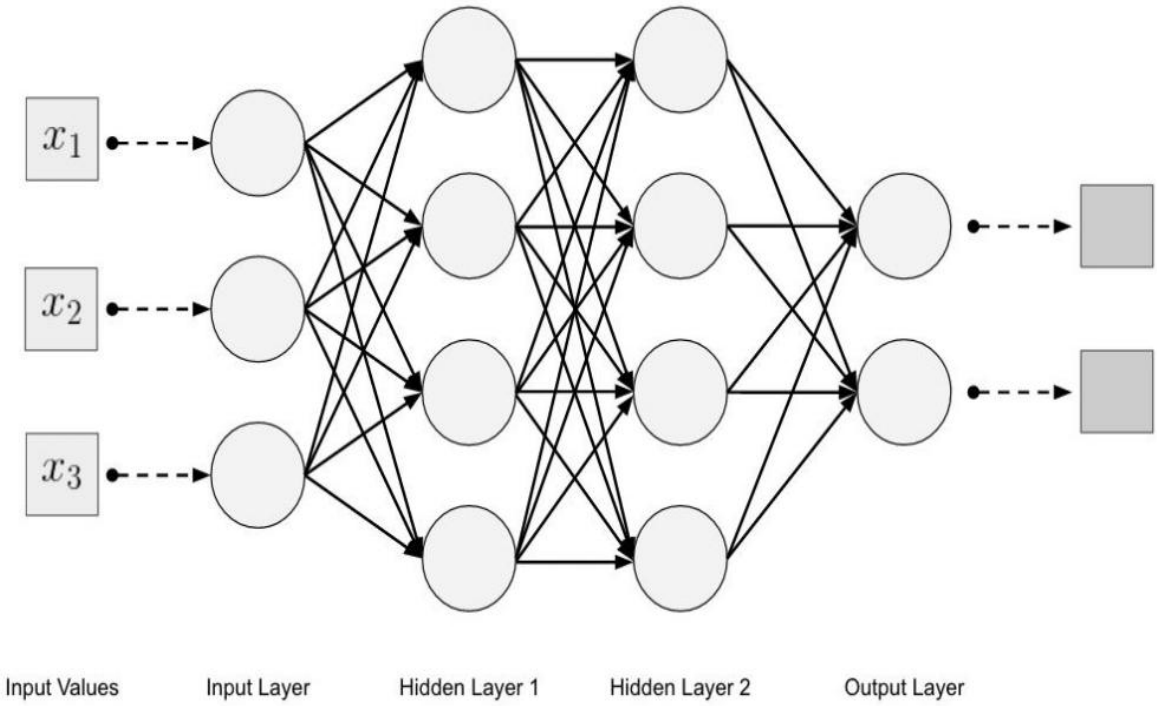


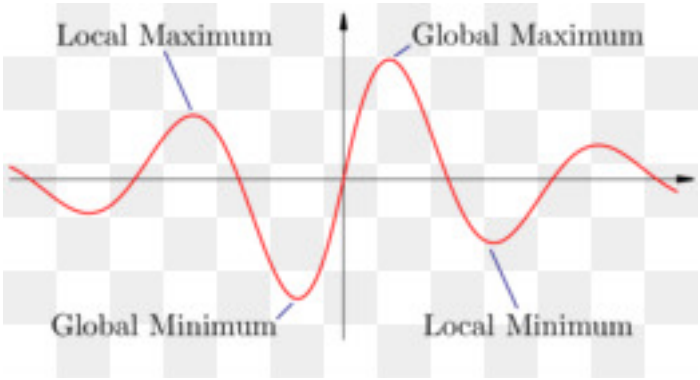
Рисунок 1.17 – приклад багатошарової мережі

1.5 Проблема локального оптимуму при навчанні

Навчання мережі з 1 прихованим шаром із застосуванням контрольованого навчання зазвичай призводить до наближення параметрів до відповідних значень. Але при навчанні глибокої мережі, це працює набагато рідше. Зокрема, навчання нейронної мережі із застосуванням

навчання з учителем включає в себе вирішення проблеми неопукої оптмізації.

У глибокої нейронної мережі можуть з’я витися локальні оптимуми саме у великій кількості, у такому випадку навчання з градієнтним спуском може перестати працювати. Приклад такої ситуації бачимо (рисунок 1.18).



Русунок 1.18 – приклад проблемної для навчання функції

1.6 Градієнтна дифузія

Якщо використовувати метод оберненого розповсюдження помилки то градієнти котрі розповсюджуються від останнього шару(вихідного) до попередніх шарів нейронної мережі, дуже швидко зменшуються зі збільшенням шарів мережі. Це потрібно для визначення похідних у методі оберненого розповсюдження помилки. Звідси виходить що ранні шари навчаються повільно по відношенню до наступних, це слідує з того що ваги ранніх шарів змінюються повільніше. Тож при використанні методу градієнтного спуску виникає проблема котру назвали – дифузією градієнтів(diffusion of gradients).

Зникаючим градієнтом називають проблему котра виникає у випадку навчання штучних нейромереж із застосуванням зворотнього поширення помилки і градієнта. Використовуючи такі методи, вагові коефіцієнти

нейромережі змінюються пропорційно до градієнту функції помилки відносно вагових конефіцієнтів у даний момент(на кожній ітерації). Оскільки у певних функцій активації наприклад гіперболічного тангенса, градієнт лежіть у проміжку -1 та 1, а метод зворотнього поширення помилки обчислює їх ланцюгом, то ми маємо що в n - шаровій мережі градієнт швидко зменшується(експоненційно) разом з n, а передні шари навчаються повільно. А якщо використовувати функції активації с великими значеннями похідних то можемо наткнутися на exploding gradient problem. Вибухові градієнти - це проблема, коли великі градієнти помилок накопичуються і призводять до дуже великих оновлень ваг моделей нейронної мережі під час тренувань. В крайньому випадку, значення ваг можуть стати настільки великими, що переповнювати і призводити до значень NaN.

Приведемо певні ознаки що наша мережа потерпає від вибухового градієнту

- Модель не може правильно обробляти дані для навчання
- Модель є нестабільною – маємо великі втрати з кожним оновленням
- Під час тренування значення переходять в Nan

Вирішення проблеми зі зникаючим градієнтом може бути наступним.

Першим рішенням може бути використання багаторівневої ієрархії – це коли шар навчається методом без вчителя, але потім його значення корегується методом оберненого поширення помилки, так кожен шар вивчає певне представлення про спостереження, котре потім передається на наступний шар.

Другим рішенням є використання Residual Networks(ResNets скорочено), є насправді одним з найефективніших методів для того щоб вирішити дану проблему. Мережа з більшою кількістю шарів(більш глибока мережа) нажаль має більшу помилку при навчанні ніж мережа з малою кількістю шарів. Тому розробники з компанії Microsoft помітили що при умовному ділені мережі на частини, наприклад по три шари мережі, та

передача даних за домогою зв’язку швидкого доступу – цей зв'язок пропускає один чи більше шарів та додають виходи попередньої частини до виходу поточної. Цей метод досить просто дозволяє оптимізувати мережу коли її глибина збільшується., також вона зі збільшенням глибини дозволяє збільшити точність, отримати такі результати дуже складно у інших мережах. Ілюстрацію роботи можемо бачити на (рисунок 1.19).

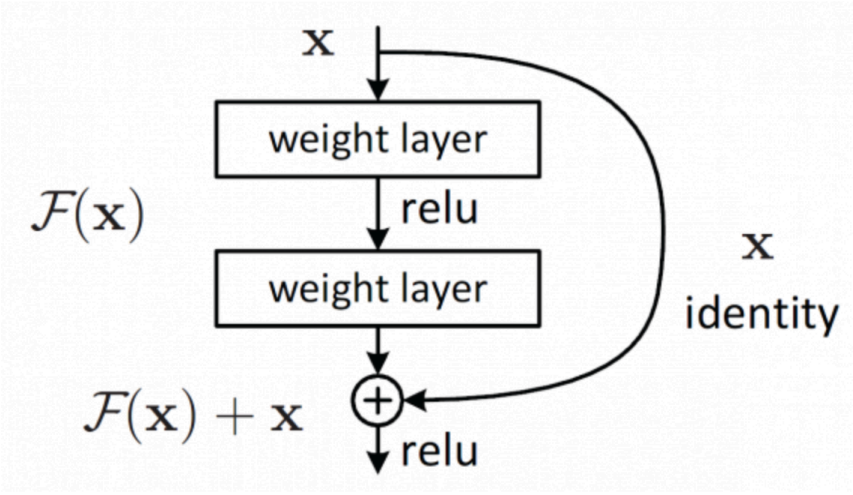


Рисунок 1.19 – зв'язок швидкого доступу

Третім рішенням такої проблеми є використання такий вид архітектури рекурентної нейронної мережі як – довга короткострокова пам'ять(Long Short Term networks, LSTMs). Ці штучні нейронні мережі були представлені Хорхайтером та Шмідхубером у 1197 році та у майбутньому дуже популяризовані і використовувані у роботі та для вирішення задач. Суть їх в тому що при розповсюдженні помилки від вихідного шару, вона не виходить с LSTM блока, вона постійно передається назад поки подібні значення не будуть відкидатися.

1.7 Висновки

У цьому розділі ми розглянули основну теорію пов’язану з нейронними мережами, а саме найпростіші складові штучної нейронної мережі, види мереж та їх архітектури, методи навчання нейромереж. Також ми ознайомились з проблемами котрі виникають при навчанні різних нейронних мереж та методи котрі дозволяють їх вирішити. Таким чином ми підготувалися щоб розглянути цільову для нас нейронну мережу CNN, аналоги для неї та порівняти їх.

2 АРІТЕКТУРА CNN ТА ТЕОРІЯ БАЄСІВСЬКОГО КЛАСИФІКАТОРА

2.1 Згорткові нейронні мережі

Останні роки штучні нейроні мережі стали більш популярними та мали великий розвиток, це призвело до проривних результатів у рішенні багатьох завдань що постали перед людством, таких як – комп'ютерний зір та розпізнавання голосу. Одним з факторів що привів до цього є нейронна мережа котра називається – згортковою. Якщо просто описати CNN то це мережа котра використовую безліч копій одного нейрона, це дозволяє працювати з великими моделями та зберігати туж кількість параметрів.

Згорткою називають операція котру можна застосувати до двох послідовностей(наприклад f і g) і буде породжена третя послідовність

$$(f * g)(c_1, c_2) = \sum f(a_1, a_2)\, g(c_1 - a_1, c_2 - a_2) \qquad (2.1)$$

де f – якась фукція,

g – якась функція,

a1, a2, c1, c2 – коефіцієнти.

Згорткова нейрона мережа є однією з архітектур штучних нейромереж, котру створювали для аналізу(розпізнавання) зображень, використовують по черзі згорткові шари, потім RELU, шари об'єднання і вихідний шар. Приклад на (рисунок 2.1).

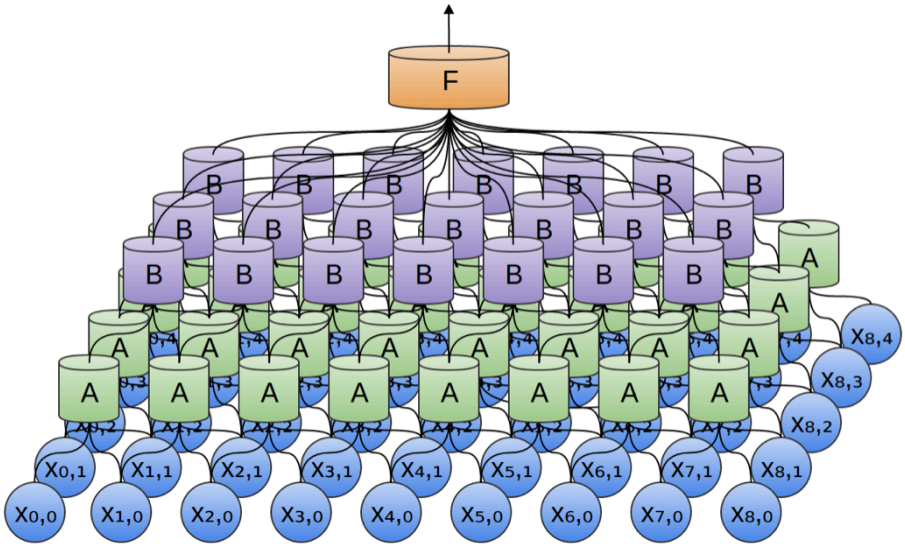


Рисунок 2.1 – 2D зображення CNN

Нейрони згорткової нейронної мережі розташовуються в 3-х вимірах, як це показано на одному з шарів. В даному прикладі червоний вхідний шар містить зображення, тому його ширина і висота визначається розмірами картинки, а глибина буде дорівнювати 3. Приклад на (рисунок 2.2).

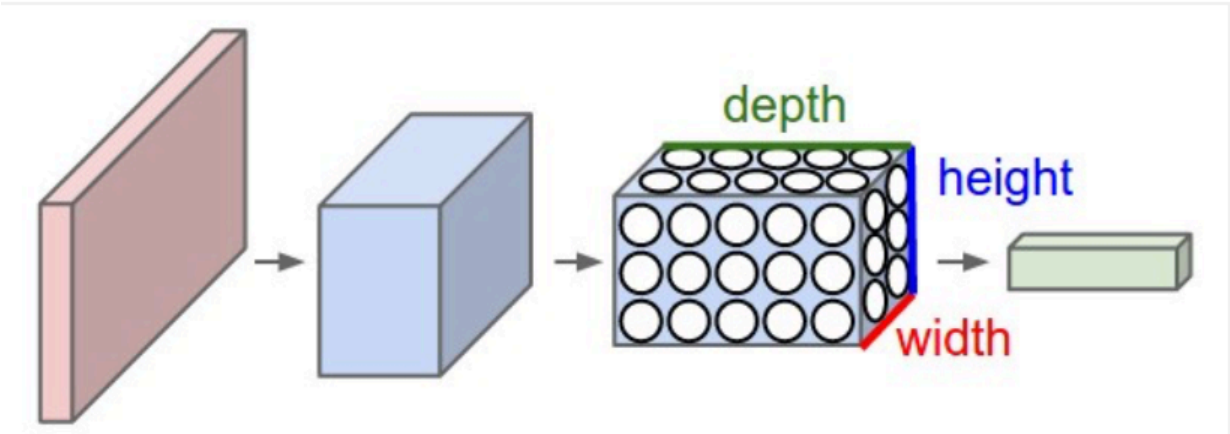


Рисунок 2.2 – 3D згортка

Для розуміння розглянемо одновимірний згортковий шар з n входами та виходом котрий представимо як F(там може бути yn вихідних нейронів). Тоді матимемо наступну схему зображену на (рисунок 2.3).

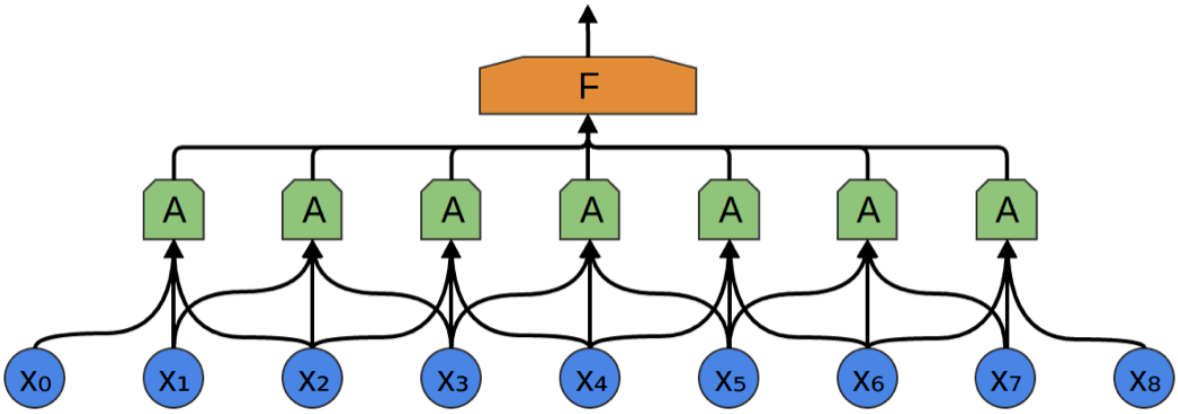


Рисунок 2.3 – Одновимірний вид шару

Раніше ми казали, що в середині згорткового шару є безліч копій одного нейрона тому на виході можемо мати однакові ваги для певних у

$$Y1 = f(W0x0+W1x1+W2x2+b) \quad (2.2)$$

$$Y2 = f(W1x1+W2x2+W3x3+b) \quad (2.3)$$

де W – ваги,

x – вхідні дані.

Матриця ваг поєднує кожен вхід з кожним нейроном. Матриця згорткового шару має особливість в тому, що в певних місцях може бути 0, адже нейрони не з’єднані з усіма можливими входами, також ваги у матриці можна зустріти декілька разів. Для операції згортки використовується матриця ваг не великого розміру котра рухається по шару котрий ми оброблюємо та після цього формує сигнал активації для нейрону наступного шару. Цю матрицю називають kernel(ядро), також для її характеристики використовують термін kernel size(розмірність ядра) – він характеризує розмірність матриці(ядра). У випадку коли ми маємо один канал то терміни фільтр та ядро мають одне й теж значення, але коли ми розглядаємо наприклад кольорове зображення де є більше одного каналу кольору то фільтр є набором ядер, кожне з них ставиться у відповідність каналу зображення.

У процесі виконання операції згортки кожний фрагмент на котрий накладається ядро множиться на матрицю згортки поелементно, а потім ці значення додаються та записуються у відповідну позицію. Таким чаном отриманий шар вказує на те – чи є певна ознака. У CNN існує багато наборів ваг, котрі формуються під час навчання нейромережі. У результаті після проходу ядром формується карта ознак і так формується певна їх кількість, як результат ми маємо багато карт ознак на одному згортковому шарі і таким чином нейронна мережа стає багатоканальною.

З шарами згортки ставлять шар pooling котрий служить для зменшення розміру карт ознак, що дає змогу зменшити кількість обчислень, праметрів мережі, а також контролювати і не дати перенавчитися. Цей шар масштабує об’єм просторово, використовуючи функцію максимуму. Береться фільтр з певним розміром, наприклад 2X2 та крок 2, та аналізуються 4 числа з котрих обирають найбільше (рисунок 2.4).

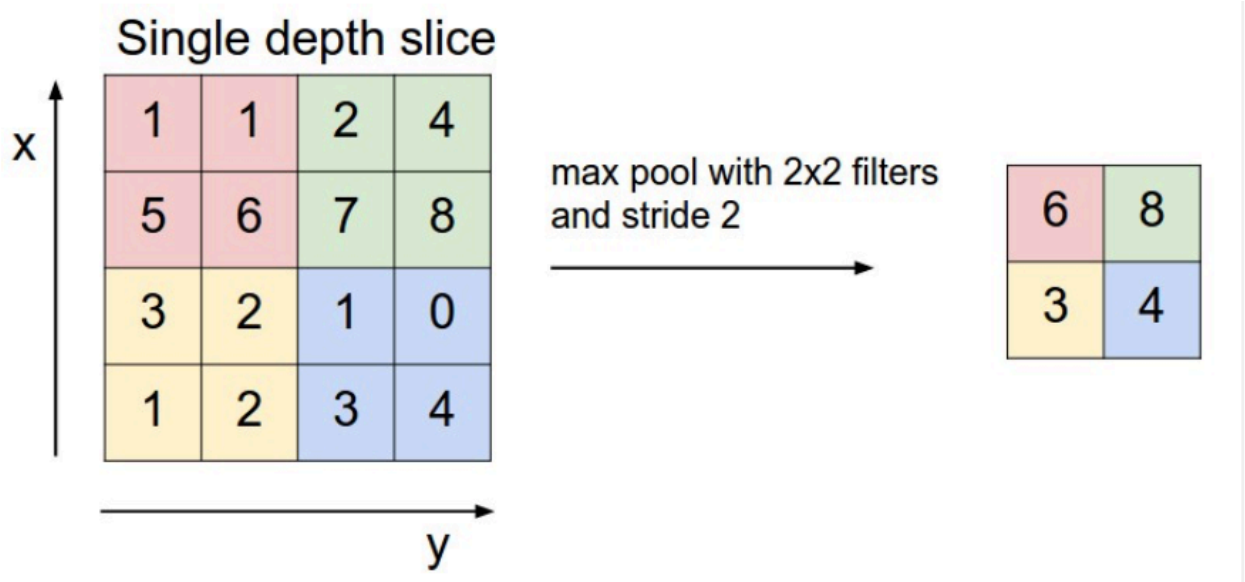


Рисунок 2.4 – приклад роботи pooling шару

Після проходження певної кількості шарів карта ознак витягується у вектор або навіть скаляр, але таких карт стає досить багато, тому далі їх передають на декілька шарів нейромережі наприклад персептрону.

Застосування згорткових нейронних мереж у роспізнаванні тональності тексту.

Спочатку потрібно виконати векторизації слів для цього потрібно ініціалізувати шар котрий називається embedding. Цей шар використовується щоб закодувати слова цілими числами і як результат кожне слово закодоване унікальним числом, тому важливо перед початком навчання підготувати дані для навчання. Досить популярними рішеннями є використання Word2vec, Glove та FastText, один з цих інструментів ми і використаємо.

Вхідними даними нейронної мережі є матриця із зафіксованою нами висотою(позначимо n), в цій матриці кожний рядок це векторне представлення слова, висота матриці може бути задана різна, оскільки вона відображає кількість слів у тексті, зазвичай обирають кількість котра покриває майже всі тексти, у разі якщо ми покриємо не всі слова у тексті то вони можуть просто відкидатися. Для згорткових шарів обирають фільтри різних розмірів та додають певну кількість шарів для кожної розмірності, а в якості функції активації часто використовують функцію ReLU, котру ми також будемо використовувати. Після проходження всіх згорткових шарів використовується шари субдескритизації до карт ознак котрі ми отримали, таким чином ми можемо лишити тільки значущі ознаки і відкинути те що не потрібно, зменшивши розмірність карт (рисунок 2.5).

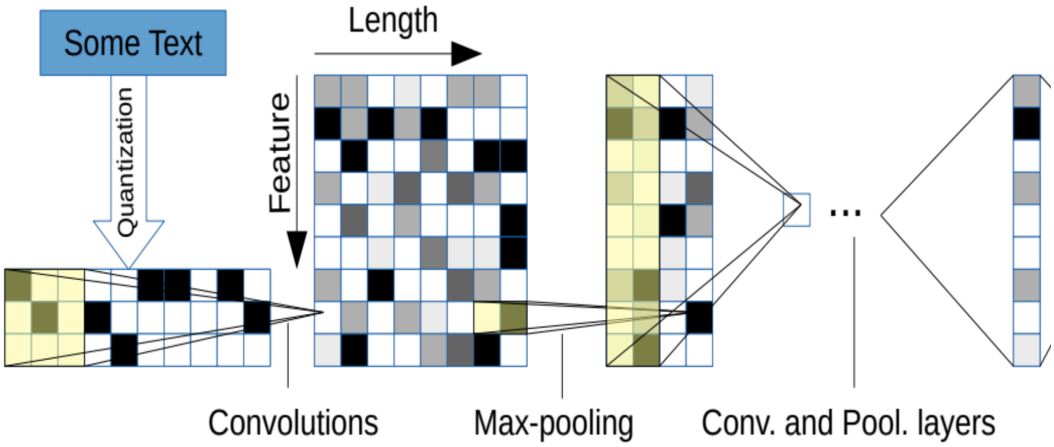


Рисунок 2.5 – Схема частини мережі з підготовкою тексту та згортковими шарами і субдекритизації

На наступному етапі зазвичай відбувається поєднання карт у один вектор ознак, це відбувається за допомогою шару поєднання з котрого карта ознак потрапляє на повнозв'язний шар після котрого карта ознак потрапляє на вихідний шар нейронної мережі з певною функцію активації. Раніше було зазначено що нейронні мережі можуть не тільки погано навчитися, але й можуть перенавчитися, окільки конволюційні нейронні мережі не вийняток то використовують dropout з певною заданою ймовірністю це дуже допомагає запобігти перенавнчаню, цю регуляцію додають перед прихованим повнозв'язним шаром (рисунок 2.6).

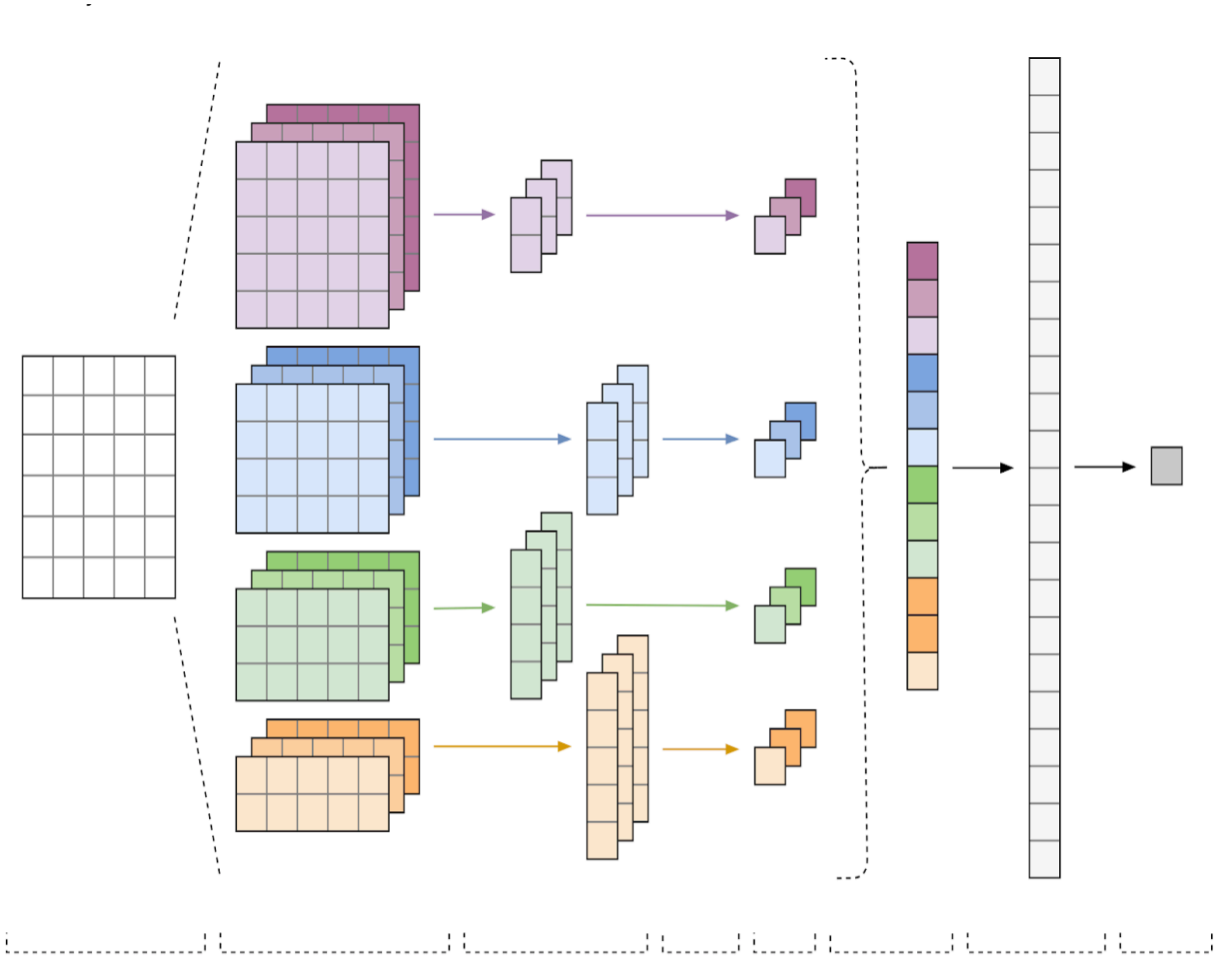


Рисунок 2.6 – Приклад побудованої нейромережі CNN для роботи з текстом

Також потрібно якимось чином оцінювати роботу цього класифікатора, для цього використовують певні метрики, котрі дозволяють оцінити оптимальність роботи нейронної мережі.

Першою такою метрикою є точність(precision), вона використовується як самостійна метрика оцінки алгоритмів так і для знаходження інших метрик оцінки мережі. Точність це величина котра відображає відношення кількості текстів що належать до певного класу до текстів що мережа віднесла до цього класу. Розглянемо приклад коли мережа з 10 речень визначила, що два з них негативні, але насправді одне з цих двох речень є позитивним за емоційним забарвленням, тоді точність мережі ½ (одне з обраних речень є дійсно негативним, а всього мережа визначила два речення як негативні). Введемо позначення:

- N_1 – Кількість елементів котрі система віднесла до правильного класу
- N_2 – Кількість елементів котрі система віднесла до класу

Тоді точність можна визначити наступною формулою:

$$\text{Precision} = \frac{N_1}{N_2} \tag{2.4}$$

де N_1 – кількість вірних елементів,

N_2 – все що віднесено до класу.

Другою метрикою є повнота(reccal), вона також може бути використана як самостійна метрика, або у підрахунку більш складних метрик таких як F-міра та R-точність. Вона відображає відношення кількості знайдених нейронною мережею текстів одного класу до всієї кількості текстів класу. Повертаючись до прикладу з 10 реченнями, додамо що з них 4 це негативні, тому повнота дорівнюватиме ¼ . Ведемо позначення:

- N_1 – Кількість елементів котрі система віднесла до правильного класу

- N_3 – Кількість елементів у класі

$$\text{Reccal} = \frac{N_1}{N_3} \tag{2.5}$$

де N_1 – кількість вірних елементів,

N_3 – все що у класі.

Третім параметром є F- міра, вона представляє гармонічне середнє значення між повнотою та точністю і вона також прямує до нуля якщо ті два параметри теж прямують до нуля. Виникає питання – навіщо ми вводимо ще одну метрику, а справа в тому що у реальних задач максимальна повнота не може бути досягнута одночасно з максимальною точністю і тому треба знайти певний компроміс, саме тому була створена ця метрика котра поєднує два попередні параметри. За нею часто приймають рішення який варіант обрати для реалізації. Формула при котрій пріоритет між повнотою та точністю є однаковою:

$$\text{F} = 2 * \frac{\text{Precision} * \text{Reccal}}{\text{Precision} + \text{Recca}} \tag{2.6}$$

де Precision – точність,

Reccal – повнота.

У випадку коли ми хочемо віддати перевагу одній з метрик то вводять параметр β. Якщо він від 0 до 1 то перевага віддається точності, а якщо більше 1 то повноті.

$$\text{F} = (\beta + 1) * \frac{\text{Precision} * \text{Reccal}}{\beta^2 * \text{Precision} + \text{Recca}} \tag{2.7}$$

де Precision – точність,

Reccal – повнота.

Формально цей параметр є дуже гарним для оцінки якості роботи нейронної мережі, хоча б тому що він поєднує два параметри та спрощує порівняння, як результат спрощує процес прийняття рішення.

На перший погляд коновлюційна нейронна мережа не здається найкращим варіантом для роспізнавання тексту та його емоційного забарвлення, адже коли на зображені пікселі поряд беруться фільтром то це логічно, адже вони частина одного зображення і пов’язані між собою, в той час як слова у реченні можуть бути частинами фраз котрі розділені іншими словами. Але як ми розглянемо далі ця мережа показує гарні результати і навіть кращі ніж конкуруючі з нею мережі.

Гіперпараметри мережі та архітектура:

Першим гіперпараметром мережі є глибина, ця величина є еквівалентною до кількості фільтрів котрі ми хочемо використати(або маємо). Наприклад при роспізнаванні зображень RGB ми маємо три так званих канали, а в обробці тексту це можуть бути різні представлення слів, наприклад перефразовані речення, інша мова і тому подібне.

Другим гіперпараметром є доповнення нулями, що дає змогу контролювати просторові розміри вхідних данних, розмір цього доповнення і є гіперпараметром. Найчастіше це використовують для того щоб зробити значення висоти та ширини однаковими для вхідних даних, або для того щоб не отримати вузьку згортку то можна використати доповнення нулями.

Третім гіперпараметром є розмір кроку зсуву фільтру на кожному кроці, коли крок наприклад дорівнює одиниці то у випадку із зображеннями ми зсуваємо фільтр на один піксель в бік, коли значення два і так далі то ми перестрибуємо два чи більше пікселі, але найчастіше використовують крок один чи два. Також варто зазначити, що більше крок ми обираємо, ти менше отримаємо розмір вихідної матриці згортки, цікаво те що якщо взяти великий крок то поведінка мережі може нагадувати поведінку рекурсивної мережі.

Типова для такої мережі структура це однонапрямлена мережа котра для навчання використовує зазвичай метод оберненого поширення

помилки(або певні модифікації типу Adam, Adadelta, rmsPRoP), також помітимо що їй характерна велика кількість шарів. Приклад на (рисунок 2.7).

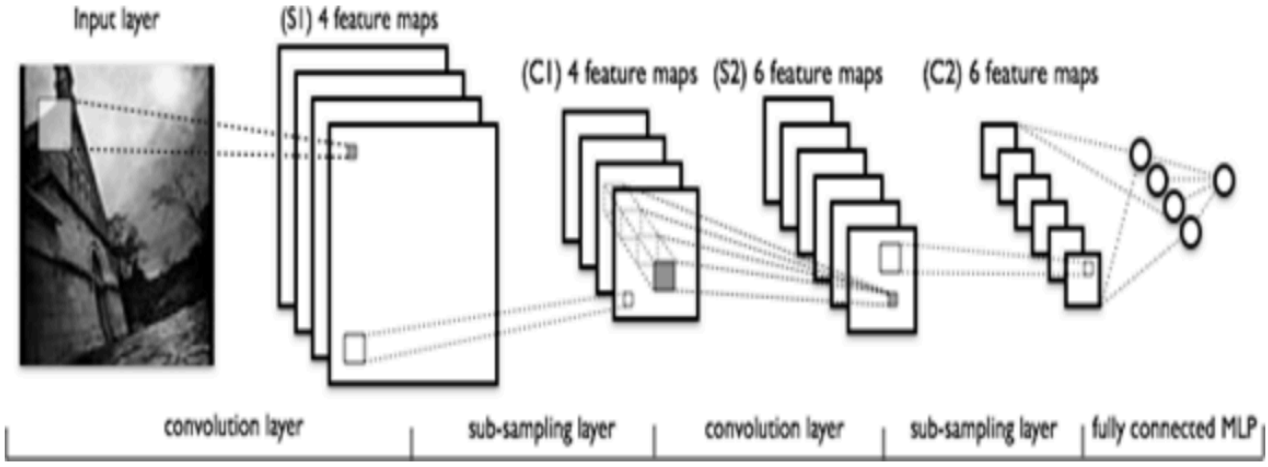


Рисунок 2.7 – архітектура мережі

2.2 Метод Adam

Вибір алгоритму для оптимізації системи є дуже важливим питанням, адже це може певним чином покращити результати навчання, та час навчання мережі.

Метод Адама є досить новим, та отримав досить велику популярність при навчанні глибоких мереж у сферах комп'ютерного зору та роспізнавання мови та тексту. Також варто зазначити що цей метод є модифікацією методу стахестичного градієнтного спуску, тож часто використовується замість нього для ітеративного корегування ваг мережі на основі вхідних даних.

Розглянемо чому саме обирають цей алгоритм, для цього наведемо його переваги котрі надали автори методу:

- Метод добре працює із задачами котрі вимагають великих об'ємів даних;

- Є досить простим та не привередливим до сервера на котрому відбувається навчання, хоча б тому що не потребує багато пам’яті;
- Гіперпараметри методу є досить простими для розуміння та часто не потребують важкого та довгого налаштування;
- Відносно інших модифікацій є досить простим для реалізації та не потребує довгого написання коду;
- Має велику обчислювальну ефективність;

Порівнюючи метод Adam з його початковою версією – методом стохастичного градієнтного спуску, то перший підтримує постійну швидкість навчання для всіх регуляцій ваг, в другому випадку було поєднано найкращі сторони двох інших модифікацій першого і метод адаптує швидкість навчання для кожного з параметрів використовуючи першого та другого моментів градієнту. Виходить так що Adam на відміну від методу RMSProP використовує не тільки середнє першого моменту, але й середнє другого моменту градієнта – нецентрована дисперсія.

Для демонстрації можливостей цього методу було проведено навчання багатошарового персептрона і порівняно результати(training cost) з іншими методами. Приклад на (рисунок 2.8).

Далі розглянемо параметри котрі має даний метод, вони мають певні стандартні значення у популярних бібліотеках наприклад мови Python, таких як Tensorflow, Keras(надбудова Tensorflow), Blocks.

- Параметр α - це значення задає швидкість навчання, або розмір кроку з котрим оновлюються ваги.
- Параметр $\beta 1$ – це показник експоненційного затухання першого моменту
- Параметр $\beta 2$ – цей параметр має такий самий сенс як і попередній але для другого моменту, це значення може бути близьким до 1 або навпаки дуже малим значенням.

Всі ці параметри мають певні стандартні та рекомендовані значення у вище наведених бібліотеках для спрощення роботи з ними.

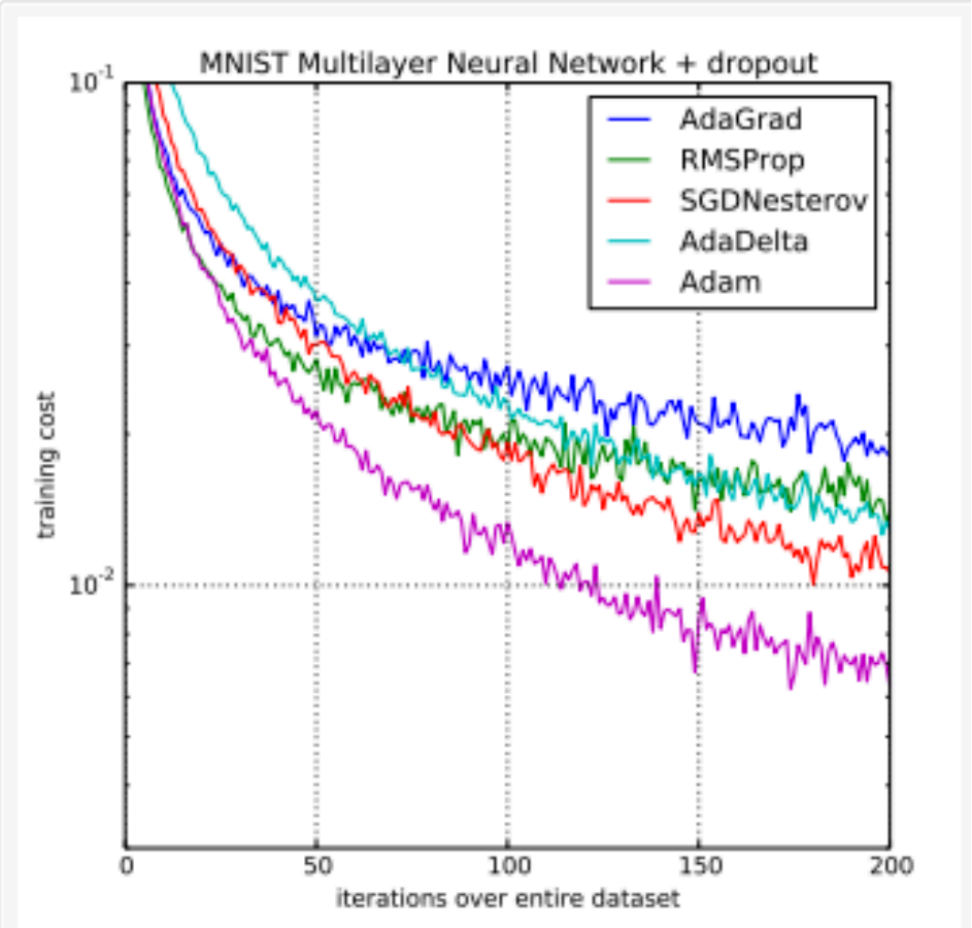


Рисунок 2.8 – Графік порівняння методу Адам з іншими

2.2 Наївний баєсівський класифікатор

Для вирішення задач аналізу емоційного забарвлення тексту у якості базового рішення використовую наївний баєсівський класифікатор. Головною ідеєю цього алгоритму є класифікувати текст використовуючи імовірність класів котрі надані текстам, використовуючи залежні ймовірності класів та самих слів. Тож якщо розглянути певне речення “T”, нехай це буде коментар у соціальній мережі та маємо певні класи нехай “K”, то ми маємо знайти такий клас “k” щоб ймовірність входження речення “T” в цей клас була найбільшою серед всіх в “K”. Запишемо формулу котра б відображала наш приклад:

$$k = \text{argmax}P(K/T) \tag{2.8}$$

де P(K/T) – відносна ймовірність.

Оскільки знайти ймовірність К відносно Т не надто проста задача, то використовую теорему Баєса:

$$P(K/T) = \frac{P(T/K) * P(K)}{P(T)} \tag{2.9}$$

де P(T) – це константа за умови що значення змінних ознак відомі,

P(K) – це йморівність.

Тоді можемо переписати формулу наступним чином:

$$P(K/T) = \frac{P(t1, t2, .../K) * P(K)}{P(t1, t2, ...)} \tag{2.10}$$

де P(t) – це константа за умови що значення змінних ознак відомі,

P(K) – це йморівність.

Але це все рівно не дає змоги порахувати те що потребуємо , тоді ми робимо наївне припущення(звідси назва) що Т залежить тільки від К та не впливають один на одного, це дуже спрошує те що ми намагаємось зробити, але враховуючи що це базове рішення та те що зазвичай воно працює то можемо це робити. Тому кінцева формула буде виглядати ось так:

$$k = \text{argmax}P(k)\prod P(t_i/k) \tag{2.11}$$

де P(t) – це константа за умови що значення змінних ознак відомі,

$P(t_i/k)$ – відносі ймовірності.

Як результат маємо знайти два параметри у формулі, в цьому і заключається тренування даного класифікатору.

2.3 Висновки

У цьому розділі ми розглянули архітектуру та побудову, а також принцип конволюційної нейронної мережі. Також ми ознайомились з будовою кожного шару мережі, а також їх функціональними особливостями та задачами. Визначили які можуть бути гіперпарамери мережі та в яких межах вони лежать та на що впливають у мережі. Розглянули використання конволюційних нейронних мереж для роботи з текстами, та певні особливості навчання таких мереж, та методи котрі для цього використовують. Описали метод Adam – його перваги над іншими модифікаціями, привели графік порівняння при навчанні персептрону з використанням цього методу. Розглянули його основні параметри, та виявили рекомендовані та можливі значення цих параметрів. Також описали в яких бібліотеках мови програмування Python є реалізований метод Adam та те що в ньому є налаштовані параметри для спрощення роботи з ним.

Описали принцип роботи наївного баєсівського класифікатора, та те в чому сенс його навчання. Привели формули та теорему на котрій будується вся теорія для баєсівського класифікатора.

3 ОПИС МЕРЕЖІ ТА ПОРІВНЯННЯ І АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вступ

У цьому розділі ми розглянемо побудову нашої нейронної мережі CNN, а також підхід до її навчання. Підготуємо дані для навчання мережі та поділимо їх на навчальну вибірку та тестову, за котрою ми зможемо оцінити ефективність роботи штучної нейронної мереж та порівняти її ефективність з базовим рішенням у сфері розпізнавання емоційного забарвлення текстів це наївний баєсівський класифікатор.

Розглянемо процес навчання та інструменти котрі для цього будуть використовуватись, такі як бібліотеки мови програмування Python 3 їх особливості, переваги та недоліки

3.2 Вимоги до технічних та програмних інструментів

Розробка, планування архітектури з парамтерами та навчання нейронної мережі вимагає певного спеціального програмного забезпечення, а також вибору бібліотек котрі полегшують задачу написання мережі та її навчання. Тому для реалізації конволюційної мережі для аналізу тональності текстів було використано наступні апаратні засоби:

- Мова програмування Python версії 3.7;
- Операційна система MacOS 10.14.6;
- Середовище для розробки це Jupyter Notebook, також можна використовувати можливості консолі для перевірки роботи блоків коду;
- Процесор Intel Core i5 2,9 GHz, графічний процесор Intel Iris Graphics 6100

За такої конфігурації комп’ютера навчання одному з етапів котрі ми далі розглянемо відбувається довго та надто сильно навантажує систему, тому для економії часу та підвищення ефективності розробки краще використовувати хмарні ресурси такі як – Amazon Web Services, Google Cloud, Microsoft Azure та Alibaba. Ці сервіси надають сконфігуровані сервери параметри котрих можна задати перед створенням, що дає змогу оптимізувати витрати коштів та пришвидшити навчання.

Також для проведення експериментів можна використати потужності ресурсу Kaggle котрий надає можливість створювати, тренувати та тестувати роботу нейронних мереж з використанням їх середовища розробки Jupyter Notebook котрий базується на серверах Kaggle. Оскільки CNN для навчання використовує GPU, то це оптимальний варіант для проведення експериментів над мережею та для порівняння з іншими.

Також для реалізації нейронної мережі, наївного баєсівського класифікатора та підготовки даних використовуються бібліотеки котрі мають певні підготовлені функції котрі спрощують процес розробки та мають певні задані та перевірені стандартні параметри. Розглянемо їх нижче:

- Keras – є у прямому сенсі надбудовою над бібліотекою TensorFlow, котра містить готові функції щоб будувати нейронну мережу блоками та явно здавати параметри, що робить написання мережі.
- Re – ця бібліотека використовується на написання регулярних виразів у програмі на мові Python ми будемо її використовувати для підготовки даних до навчання.
- Sclearn – ця бібліотека використовується для вирішення певних задач машинного навчання, ми використаємо з неї дуже зручну функцію котра ділить dataset на частини випадковим чином - тренувальну та перевірочну.
- Sqlite3 – ця бібліотека використовується для роботи з базами даних sqlite, ми скористаємось нею для того щоб записати в базу даних.

- Logging – ця бібліотека використовується для записування та візуалізації того що відбувається при виконанні програми.
- Multiprocessing – ця бібліотека досить спрощує життя та пришвидшує роботу програми, адже вона дозволяє працювати с багатопоточністю, а модуль бере на себе цю задачу.
- Gensim – це бібліотека що дає змогу працювати з мовою, а також з моделями такими як Word2vec, FastText. Якщо більш точно то використовується у задачах визначення класу до якого належить текст.

3.3 Підготовка даних

Для того щоб працювати з даними, спочатку потрібно сформувати вибірку, а також підготувати для передачі нейронній мережі на навчання та перевірку на тестовій вибірці.

Оскільки в наш час є дуже поширеним та популярним вираження своїх вражень, схвалень чи навпаки висловлення невдоволень або надання рекомендацій через коментарі у соціальних мережах, то не будемо відходити від трендів та генерувати штучну вибірку. Для навчання та перевірки роботи конволюційної нейронної мережі та наївного баєсівського класифікатора візьмемо зібрану базу даних у форматі CSV, котра складаєтья з коментарів та постів у соціальній мережі Twitter. Таким чином ми отримали для навчання мережі 127 000 розмічених текстів для навчання та тестування мережі. У цю вибірку входить 115 000 позитивних текстів та 112 000 негативних. Під розміченими текстами ми розумієм, що у відповідність кожному тексту поставлено значення 0 або 1 – негативний настрій чи позитивний. Це саме дає змогу нам перевіряти правильність роботи штучної нейронної мережі, адже ця вибірка поділена на дві частини у співвідношенні 4 до 1, де навчальна вибірка більша за тестову. Також варто зазначити що у нашій вибірці не має

малоінформативних речень(так званих постів) котрі містять до 40 символів, також емоційно нейтральних та речень що повторюються.

Для навчання Word2Vec моделі котра визначає певний симантичний зв'язок між словами, виходячи з контексту у котрому вона їх зустрічає ми використали 17,5 млн текстів. Це навчання дає змогу отримати треновану модель для використання її у embedding шарі. Тому нам потрібно підготувати ці дані для передачі їх у навчання моделі.

Для того щоб почати працювати з даними потрібно виконати певні підготовчі дії для того щоб привести дані до певних стандартів. Для цього виконаємо наступні дії:

- Замінимо всі посилання котрі є у вибірці на токен URL, для того щоб вони не потрапляли у словник та не впливали на рішення мережі;
- Замінимо всі букви верхнього регістру на букви нижнього регістру;
- У всіх текстах де з'являється нікнейм(ім'я користувача у соуіальній мережі Twitter) користувача замінюємо його на токен USER;
- Видаляємо всі знаки пунктуації котрі зустрічаються у текстах;

Нижче розглянемо приклад виправленого речення відповідно до вище наведених стандартів (рисунок 3.1):

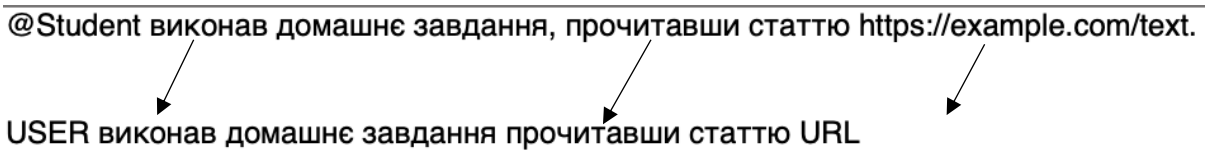


Рисунок 3.1 – приклад обробки тексту

3.4 Побудова та навчання нейронної мережі CNN та баєсівського класифікатора, а також порівняння результатів

Для побудови мережі було використано архітектуру котра описана у 2 розділі. Тому наша мережа має наступні шари:

- Embedding шар котрий дає змогу векторно представити слова векторно та надати на вхід нейронної мережі;
- Шари згортки нейронної мережі;
- 1-max-pool шар котрий дає змогу виділити основні ознаки, це також зменшує матрицю ознак;
- Шар котрий поєднує вхідні дані у один вектор ознак;
- Скритий повнозв’язний шар;
- Вихідний шар;

Спочатку ми провели навчання Word2Vec моделі на вибірці котру описали у попередньому підрозділі, це потрібно для формування нашого embedding шару, він виконує векторне представлення слів з максимально можливим урахуванням їх семантичного зв’язку для подання їх на вхід нейронної мережі. Також важливо зазначити, що всі параметри досить просто задаються завдяки існуючій бібліотеці для мови програмавання Python 3, тому ми задали параметр що дозволяє відсіяти всі слова що зустрічаються менше 3 разів у вибірці. Далі після навчання моделі ми її завантажили, ініціалізували embedding шар нулями, а потім заповнює певною кількістю що найчастіше зустрічались при навчанні моделі.

Далі потрібно визначитися з параметрами конволюціних шарів, а саме з їх кількістю та розмірністю фільтрів. Було вирішено використовувати по 10 шарів для кожної висоти фільтру, котрі були обрані такі – 2,3,4,5. Активаційна функція для них була взята ReLu адже вона найкраще себе показувала при такому застосувані(аналогом могла бути сигмоїда). Далі ми використовуємо шари субдескритизації для виділення основних ознак. Також для того щоб попередити перенавчання було використано Dropout регуляцію з імовірністю 0,2 перед повнозв’язним шаром у котрому до речі використано теж фунцію активації ReLu. Для оптимізації моделі було задіяно функцію оптимізації Адам котру ми розглядали раніше.

Процес навчання був розбитий на два етапи, перший це 10 епох з вимкненим embedding шаром на навчання, щоб мінімізувати зміни у ньому, а другий 5 епох з уже увімкненим на навчання шаром embedding.

Після перших 10 епох маємо наступні графіки точності, повноти, f-міри та функції loss, на них відображаються значення для навчальної вибірки кожної епохи та валідаційної, (рисунок 3.2 – 3.5):

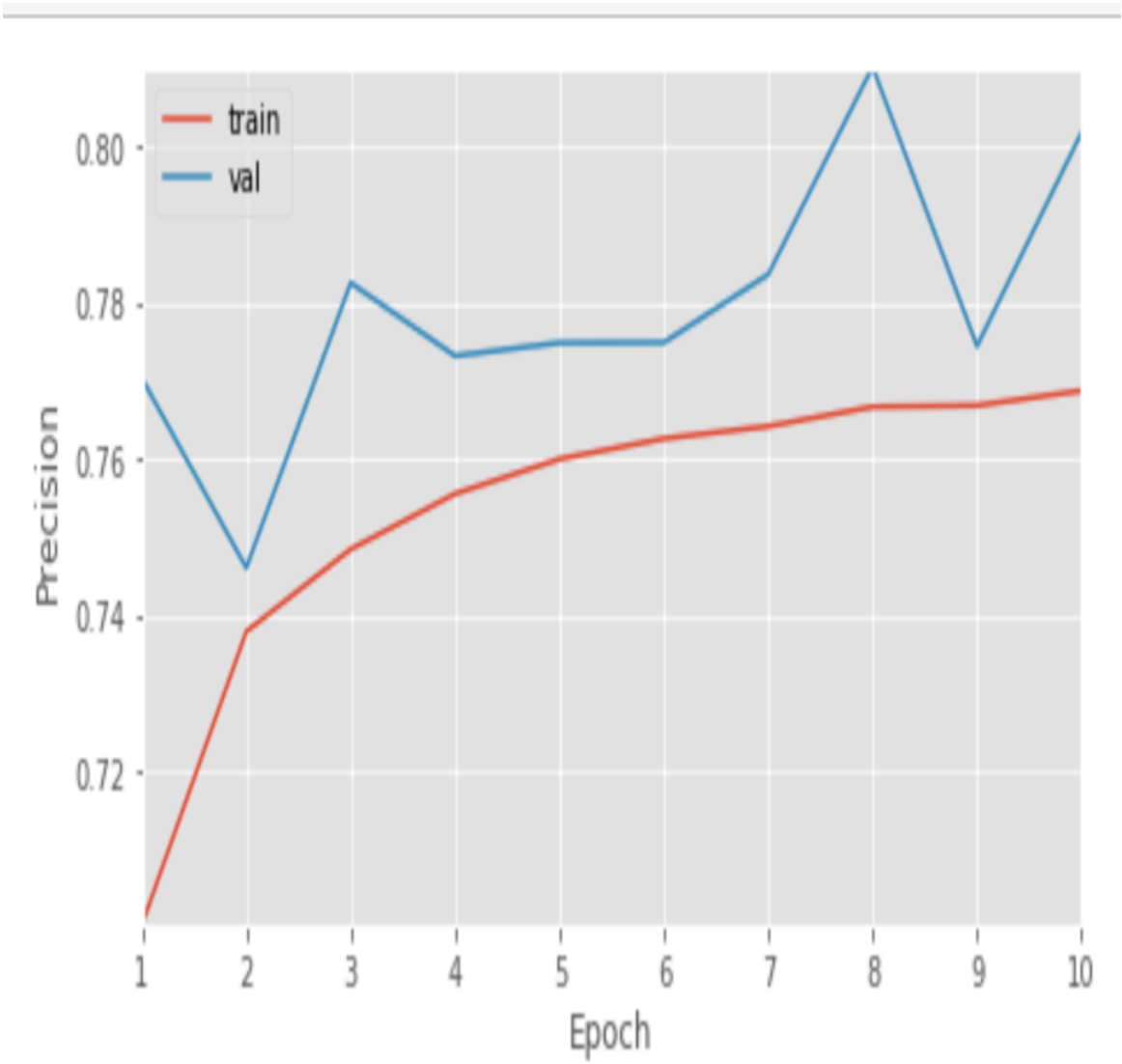


Рисунок 3.2 – Точність на кожній з епох

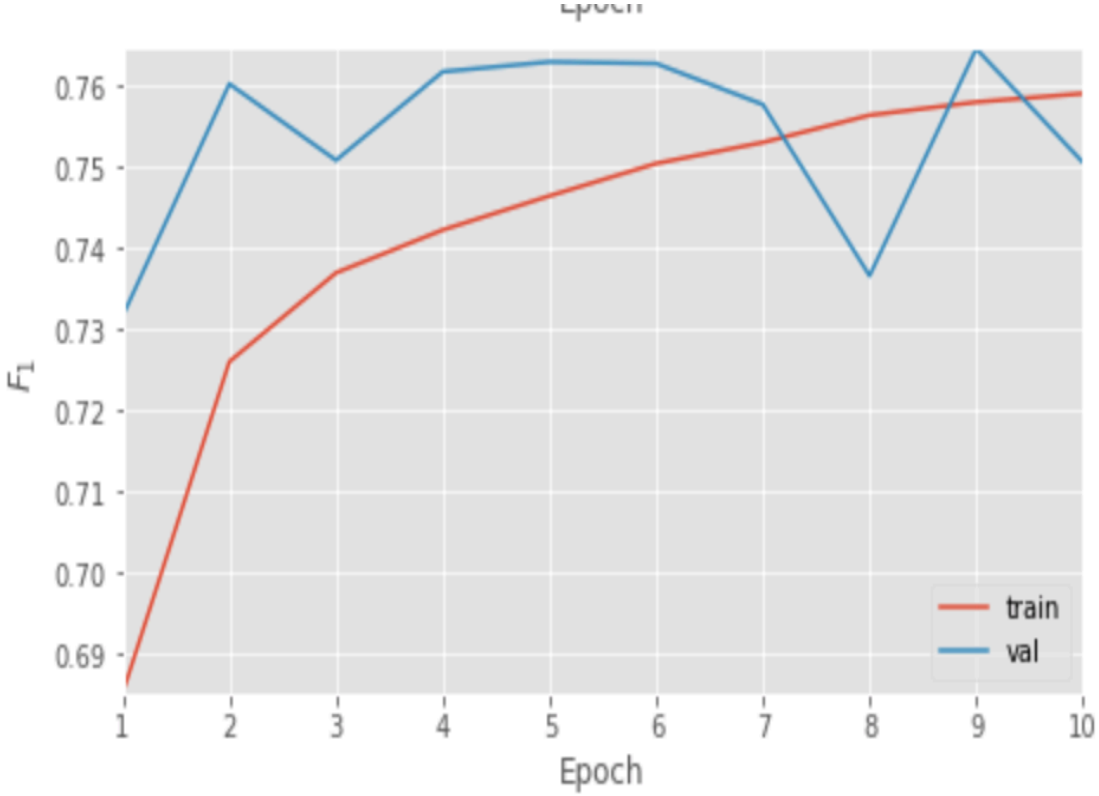


Рисунок 3.3 – F-міра на кожній з епох

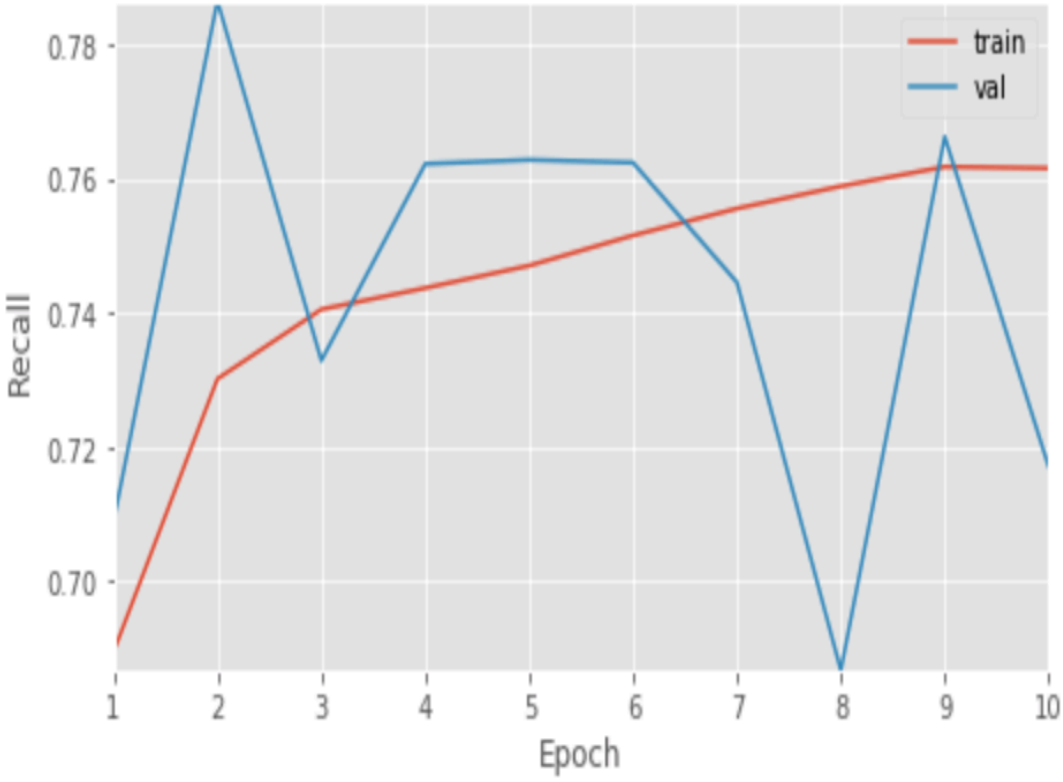


Рисунок 3.4 – повнота на кожній з епох

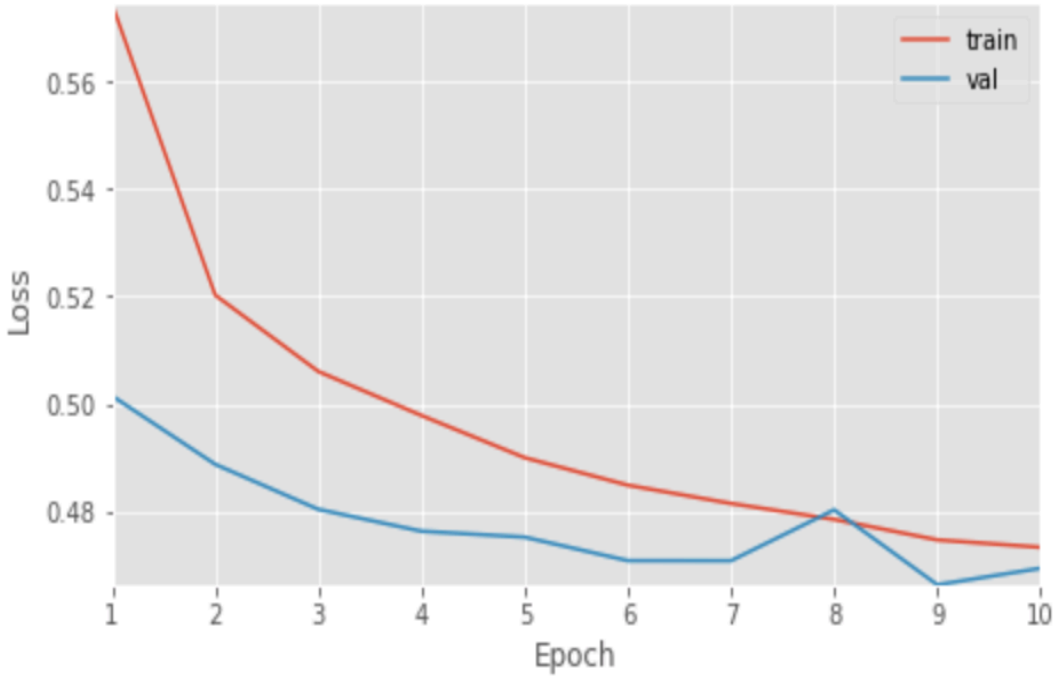


Рисунок 3.5 – loss на кожній з епох

Далі ми перевірили що ми можемо отримати на тестовій вибірці на даному етапі, (рисунок 3.6):

	precision	recall	f1-score	support
0	0.76608	0.77713	0.77156	22457
1	0.77239	0.76117	0.76674	22313

Рисунок 3.6 – параметри на тестовій вибірці

Як бачимо результати досить непогані, але після навчання 5 епох вони мають покращитись. Тому наступним кроком ми вимкнемо навчання з embedding шаром і поглянемо на результат, також перевіримо на тестовій вибірці.

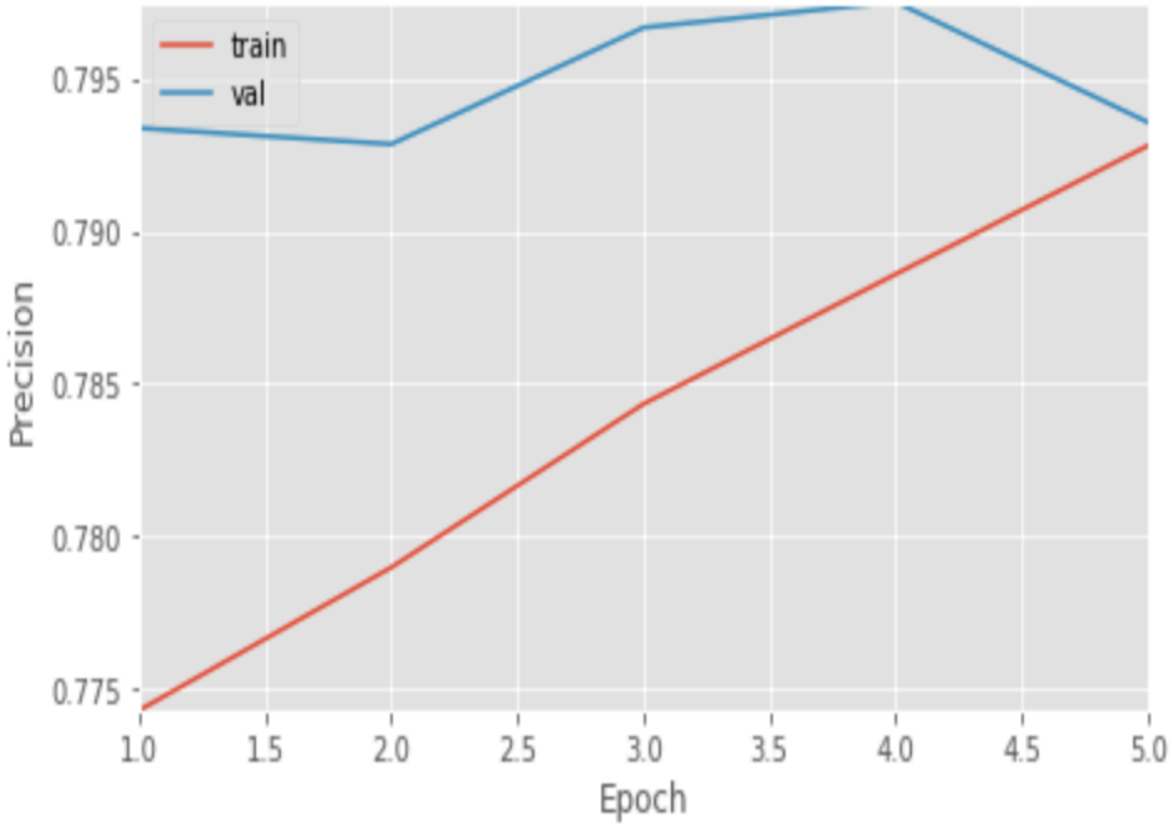


Рисунок 3.7 – точність на кожній з епох

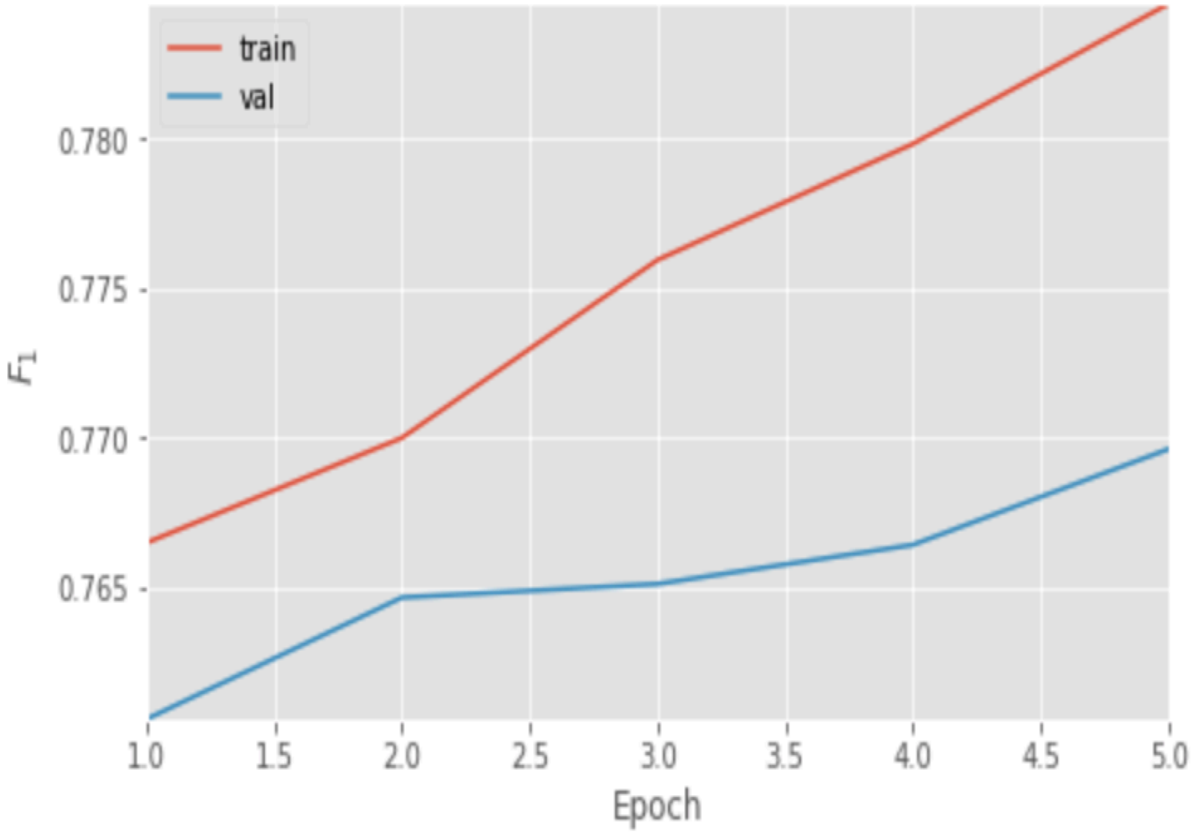


Рисунок 3.8 – F-міра на кожній з епох

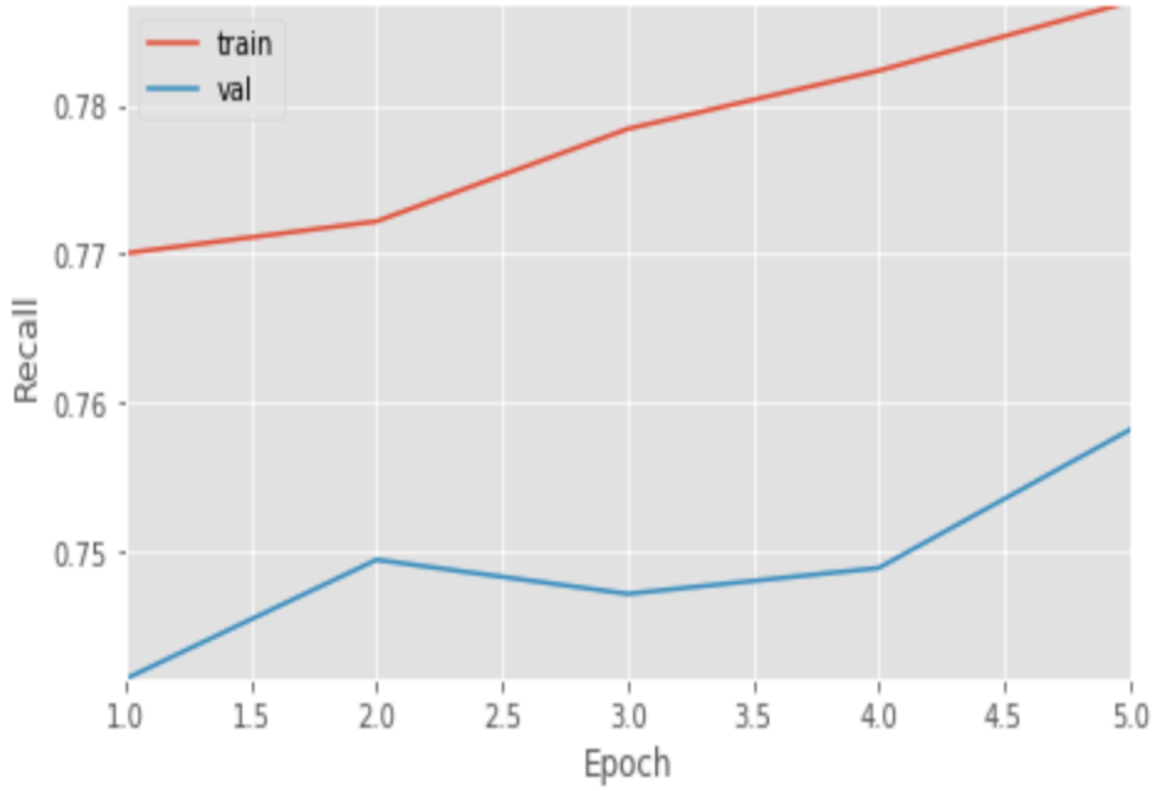


Рисунок 3.9 – повнота на кожній з епох

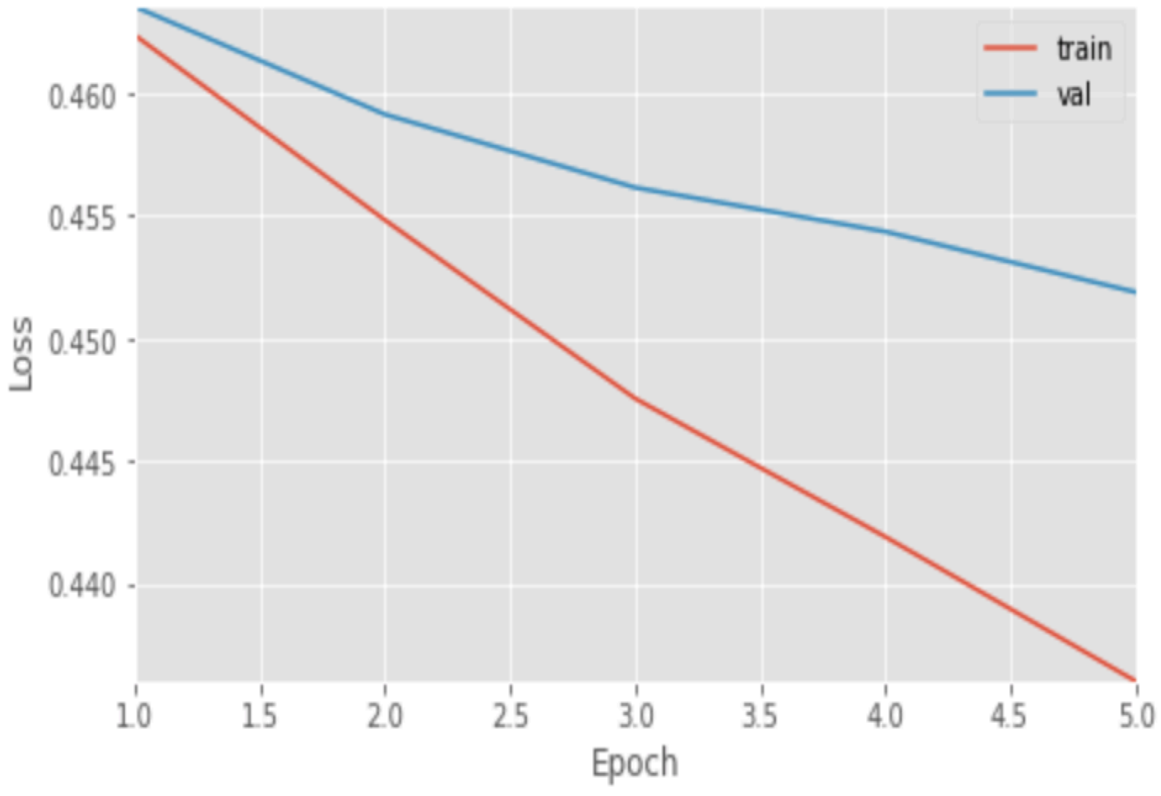


Рисунок 3.10 – loss на кожній з епох

Як бачимо з графіків що наша мережа навчається досить успішно, F – міра прийняла краще значення, а loss зменшилась, (рисунок 3.7 – 3.10). Далі мережі було знову надано тестовий набір і ось яких результатів ми дійшли, (рисунок 3.11):

	precision	recall	f1-score	support
0	0.76814	0.80371	0.78552	22457
1	0.79279	0.75584	0.77387	22313

Рисунок 3.11 – результат тестування мережі

З таблиці бачимо що результати стали кращими приблизно на 1 відсоток, що на великій вибірці охоплює велику кількість текстів, тому навчання мережі можна вважати успішним. Наступним кроком нам лишається навчити наївний баєсівський класифікатор та порівняти його результати з CNN.

Для навчання класифікатора було використано той самий набір текстів що і для CNN. Перед навчанням данні були таким самим чином оброблені та підготовані. Після навчання баєсівьского класифікатора використовуючи бібліотеку sclearn котру ми описали раніше, ми отримали аналогічний звіт по тренувальній вибірці котрий має наступний вигляд (рисунок 3.12):

	precision	recall	f1-score	support
0	0.7397	0.7941	0.7659	37078
1	0.7759	0.7183	0.7460	36792
avg / total	0.7577	0.7564	0.7560	73870

Рисунок 3.12 – результати тестування наївного баєсівського класифікатора

Порівнюючи результати отримані з використанням конволюційної нейронної мережі та результати баєсівського наївного класифікатора можемо бачити, що CNN показує кращі результати приблизно на 2,5 відсотки у кожному класі, можемо це бачити на зображені (рисунок 3.13).

Класифікатор	Precision	Recall	F1
CNN	0,7804	0,7798	0,7796
MNB	0,7577	0,7564	0,756

Рисунок 3.13 – Порівняльна таблиця

Враховуючи те що CNN можна допрацювати, протестувати різни комбінації шарів на гіперпараметри, наприклад протестувати роботу мережі з більшою кількістю прихованих шарів та нейронів у них чи змінити спосіб векторного представлення слів(адже у Word2Vec є альтернативи) або змінити кількість епох навчання та блокування embedding шару, а також те що теорія мережі CNN є зрозумілою то її використання та вдосконалення це досить об’єктивне рішення.

3.5 Висновки

У цьому розділі ми розглянули підготовку даних для надання їх нейронній мережі, та прийоми які для цього використовуються. Розглянули один із методів векторного представлення слів, а також навчання моделі Word2Vec. Побудували нейрону мережу із заданими параметрами та провели

процес навчання на навчальній вибірці, а також перевірили на тестовій. Наступним кроком було навчити на цій вибірці наївний баєсівський класифікатор, та порівняти результати з CNN.

У результаті порівняння виявилось, що нейронна мережа CNN показала результати кращі у середньому на 2,5 відсотки, що повністю виправдовує її використання, особливо враховуючи що її можна вдосконалити та оптимізувати, що покращить результати її роботи.

4 ФУНКЦІОНАЛЬНО-ВАРТІСТНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі

Проводиться оцінка основних характеристик отриманих результатів та теорії використання CNN. Для отримання практичних результатів була використана мова програмування “Python 3.7”. Середовище розробки котре використовувалось - “PyCharm CE”. Програма не залежить від того яка операційна система використовується та від апаратного забезпечення. Нижче наведені різні підходи до отримання потрібних результатів від нейронної мережі та їх аналіз з метою побудови оптимальної реалізації з технічної та економічної точки зору.

4.2 Обгрунтування функцій дослідження

Основні функції:

F_1 – об’єм розмічених даних для тренування нейронної мережі;

F_2 – вибір нейронної мережі;

F_3 – мова програмування для отримання практичних результатів;

Функція F_1 – а) 90000 розмічених речень; б)150000 розмічених текстів; в) 200000 розмічених текстів;

Функція F_2 – а) CNN мережа; б) багатошпровий персептрон;

Функція F_3 – а) Мова розробки Python 3.7; б) Мова розробки MatLab; в) Мова розробки C++;

Знизу зобразимо відповідну морфологічну карту системи:

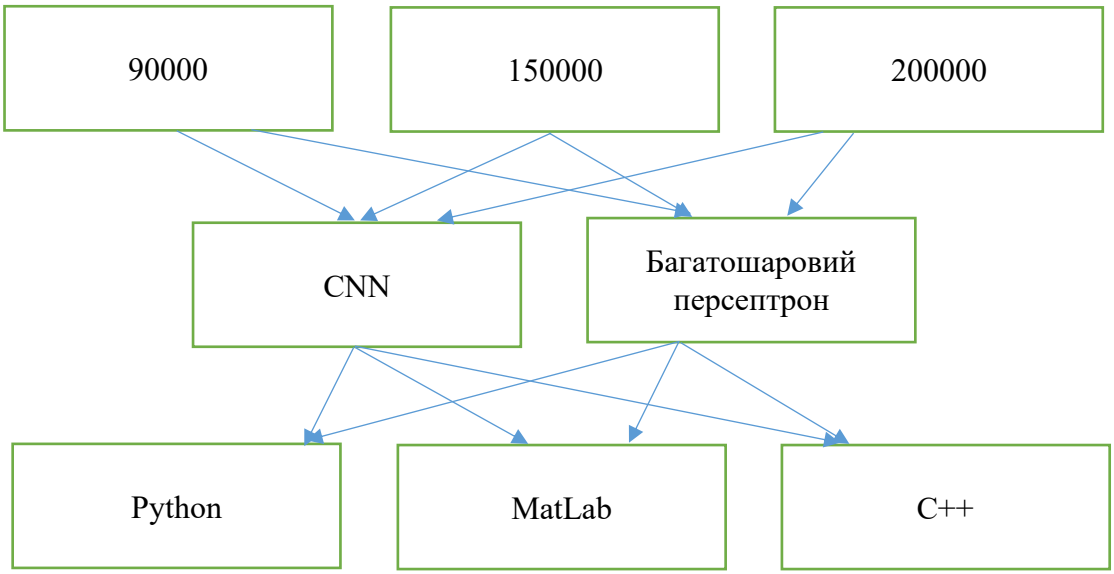


Рисунок 4.1 – морфологічна карта

Згідно з картою було побудовано позитивно-негативну матрицю

Табл. 4.1 Позитивно-негативна матриця

Порівняльна таблиця			
Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Легше отримати такий обсяг даних, та не потребує великого сховища	Цих даних мало для достатнього навчання нейромережі
F_1	Б	Нейромережі може бути достатньо такої кількості даних. Займають не багато простору на диску	Досить багато часу для їх збору

F_1	В	Вистачить для навчання мережі у будь-якому випадку	Потрібно шукати сторонню готову базу даних або дуже довго збирати свою
F_2	А	Адекватний час навчання, висока ефективність та результативність мережі	Досить складна для розуміння та побудови
F_2	Б	Проста теорія для розуміння та побудова мережі	Потребує багато часу для навчання та ресурсів комп’ютеру, а також часу для налаштувань мережі
F_3	А	Велика кількість готових та протестованих бібліотек для реалізації наших завдань, достатня кількість	Менша швидкість роботи

		літератури для вивчення	
F_3	Б	Більша лояльність до користувача	Більша потреба в ресурсах комп’ютера, менше інформації для реалізації
F_3	В	Висока швидкість виконання коду, більше простору для самостійного допрацювання готових рішень	Більш об’ємний код, менше готових бібліотек, у разі написання власного алгоритму можуть бути проблеми з тестуванням та відладжуванням коду

Тепер, за наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та недоцільності використання інших.

На основі порівняльного аналізу варіантів реалізації основних функцій по їх перевагам та недолікам можна виключити наступні варіанти F_1 А і Б , F_3 Б і В, тоді варіанти, які залишилися:

F_1 В) $->$ F_2 А -> F_3 А

F_1 В) $->$ F_2 Б -> F_3 А

Для оцінювання описаних функцій запропонована система параметрів. Опишемо цю систему.

4.3 Обґрунтування системи параметрів досліджень

Для характеризації досліджень запропонуємо такі параметри:

X1 – навантаження на процесор(у відсотках); X2 – точність роботи моделі(у відсотках, величина похибки); X3 – час навчання нейронної мережі(у годинах); X4 – час освоєння теоретичної бази мережі(у годинах);X5 – час освоєння мови програмування(у годинах); X6 – час розбору функціоналу бібліотек keras та genism(у годинах) та опис алгоритму;

F1 – (об’єм вибірки для тренування) залежить від X1 та X2 котрі представляють навантаження на процесор котре можемо собі дозволити та точність відповідно.

F2 – (тип нейронної мережі) залежить від X3 та X4 котрі представляють час навчання мережі та час освоєння теорії відповідно.

F3 – (мова програмування) залежить від X5 та X6 котрі представляють собою час освоєння мови програмування та час освоєння функціоналу її бібліотек відповідно.

Табл. 4.2 Система програмного продукту

Параметри дослідження				
Умовні позначення	Одиниці виміру	Значення параметрів		
		Гірші	Середні	Кращі
X1	%	80	50	30
X2	%	35	20	13

X3	Год	32	20	15
X4	Год	18	10	5
X5	Год	40	6	3
X6	Год	22	15	13

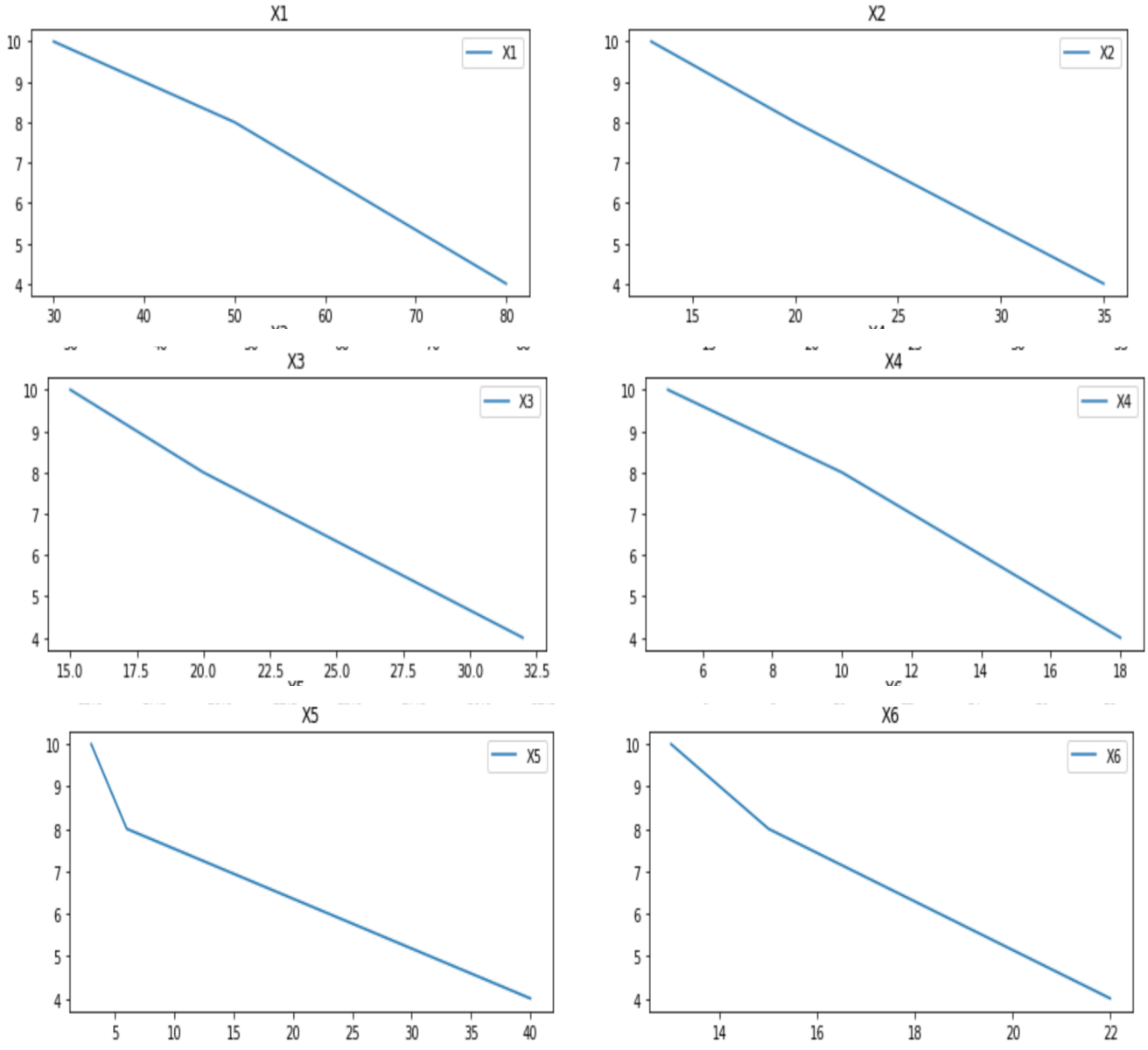


Рисунок 4.2 – графіки оцінок

Вагомість параметрів вираховується за допомогою методу попарного їх зрівняння на основі результатів ранжування експертами та порівняння параметрів попарно.

Табл. 4.3 Результат оцінки параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	ВідхиленняΔ_i	Квадрат відхилення, $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X1	1	2	1	2	1	1	1	9	-15,5	240,25
X2	3	3	2	1	2	2	2	15	-9,5	90,25
X3	2	1	4	4	3	4	3	21	-2,5	6,25
X4	4	5	3	3	4	3	4	26	0,5	0,25
X5	5	4	6	5	5	5	6	36	11,5	132,25
X6	6	6	5	6	6	6	5	40	15,5	240,25
Разом	21	21	21	21	21	21	21	147	0	709,5

Вирахуємо коефіцієнт конкординації:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 709,5}{7^2(6^3 - 6)} = 0.82 > W_k = 0.67 \quad (4.1)$$

Оскільки коефіцієнт конкординації більше нормативного то результати вважаємо достовірними.

Ранг 1 вважатимемо найнижчим, в той час як ранг 6 найвищим. Тож для найважливішого параметру експерт поставить 6, а найменш важливому 1.

Табл. 4.4 Попарне зрівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	**2**	**3**	**4**	**5**	**6**	**7**		
X1 та X2	<	<	<	>	<	<	<	<	0,5
X1 та X3	<	>	<	<	<	<	<	<	0,5

X1 та X4	<	<	<	<	<	<	<	<	0,5
X1 та X5	<	<	<	<	<	<	<	<	0,5
X1 та X6	<	<	<	<	<	<	<	<	0,5
X2 та X3	>	>	<	<	<	<	<	<	0,5
X2 та X4	<	<	<	<	<	<	<	<	0,5
X2 та X5	<	<	<	<	<	<	<	<	0,5
X2 та X6	<	<	<	<	<	<	<	<	0,5
X3 та X4	<	<	>	>	<	>	<	<	0,5
X3 та X5	<	<	<	<	<	<	<	<	0,5
X3 та X6	<	<	<	<	<	<	<	<	0,5
X4 та X5	<	>	<	<	<	<	<	<	0,5
X4 та X6	<	<	<	<	<	<	<	<	0,5
X5 та X6	<	<	>	<	<	<	>	<	0,5

Таблиця 4.5 Розрахунок вагомості параметрів

Параметри	Параметри						Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	X5	X6	bi	Kbi	bi’	Kbi’	bi’’	Kbi’’
X1	1	0,5	0,5	0,5	0,5	0,5	3,5	0,097	19,75	0,099	109,12	0,1
X2	1,5	1	0,5	0,5	0,5	0,5	4,5	0,12	23,75	0.119	130,87	0,121
X3	1,5	1,5	1	0,5	0,5	0,5	5,5	0,152	28,75	0.144	157,12	0,145
X4	1,5	1,5	1,5	1	0,5	0,5	6,5	0,180	34,75	0.175	188,87	0,174
X5	1,5	1,5	1,5	1,5	1	0,5	7,5	0,208	41,75	0.210	227,1	0,209
X6	1,5	1,5	1,5	1,5	1,5	1	8,5	0,236	49,75	0.250	272,87	0,251

Таблиця 4.6 Розрахунок рівня якості варіантів реалізації

Основна	Варіант	Параметри	Абсолютне	Бальна оцінка	Коефіцієнт	Коефіцієнт якості

функція	реалізації		значення параметру	параметру	вагомості параметру	
F_1	В	X1	35	9,5	0,1	0,95
F_1	В	X2	15	9,6	0,121	1,1616
F_2	А	X3	15	10	0,145	1,45
F_2	А	X4	6	9,6	0,174	1,67
F_2	Б	X3	17	9,2	0,145	1,334
F_2	Б	X4	14	6	0,174	1,044
F_3	А	X5	4	9,26	0,209	1,935
F_3	А	X6	18,5	6	0,251	1,506

$$K1 = 0,95+1,1616+1,45+1,67+1,935+1,506 = 8,6726 \quad (4.2)$$

$$K2 = 0,95+1,1616+1,334+1,044+11,935+1,506 = 7,9306 \quad (4.3)$$

Тож варіант, котрий передбачає використання CNN дає більший показник якості тому обираємо саме його.

4.4 Економічний аналіз варіантів розробки ПП

Обидва варіанти включають в себе три наступні етапи:

1 пошук для підготовка даних

2 навчання та корегування нейронної мережі

3 розробка програми

Для завдання 1 (Алгоритм складності 3, ступінь новизни Г, вид використаної інформації БД) $T_р$ =8, $К_п$ = 0,3, $К_{ск}$= 0,8, $К_{ст.м}$ = 1.3

$$T_1 = 8*0.3*0.8*1.3 = 2,496 \quad (4.4)$$

Для завдання 2 при реалізації першого варіанту, (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ) $T_р$ =64, $К_п$ = 2,02, К_ст = 0,65, $К_{ст.м}$ = 1,35

$$T_2 = 64 * 2,02 * 0,65 * 1,35 = 113,443 \text{ людино-днів} \quad (4.5)$$

Для завдання 2 при реалізації другого варіанту, (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ) $T_р$ =64, $К_п$ = 2,02 $К_{ск}$= 0,8, $К_{ст.м}$ = 1,55

$$T_2' = 64* 2,02 * 0,8 * 1,55 = 160,307 \text{ людино-днів} \quad (4.6)$$

Для завдання 3(Алгоритм складності 1, ступінь новизни В, вид використаної інформації БД) $T_р$ = 43, $К_п$ = 0,68, $К_{ск}$ = 0.7, $К_{ст.м}$ = 1.6

$$T_3 = 43 * 0,68 * 0,7 * 1,6 = 32,74 \text{ людино-днів} \quad (4.7)$$

$$T1 = (2,496+ 113,443 + 32,74) * 8 = 1173,432 \text{ люд-год} \quad (4.8)$$

$$T2 = (2,496+ 160,307+ 32,74) * 8 = 1564,344 \text{ люд-год} \quad (4.9)$$

У процессі розробки програмного продукту приймають участь два програміста з місячним окладом 11 000 грн.

$$C = \frac{22\,000}{2*14*9} = 87,3 \quad (4.10)$$

Зарплата п кожному варіанту:

$$C_1 = 1173{,}432 * 87{,}3 = 102\ 440 \text{ грн} \quad (4.11)$$

$$C_2 = 1564{,}344 * 87{,}3 = 136\ 567 \text{ грн} \quad (4.12)$$

На соціальний внесок відрахування становить 22%:

$$C_{1v} = 0{,}22 * 102\ 440 \text{ грн} = 22536 \text{ грн} \quad (4.13)$$

$$C_{2v} = 0{,}22 * 136\ 567 \text{ грн} = 30044 \text{ грн} \quad (4.14)$$

Визначимо витрати котра потрібна на оплату однієї машино-години. Враховуючи що заробітня плата одного програміста складає 11 000 грн з коефіцієнтом зайнятості 0,215:

$$C_г = 12 * М * К_з = 12 * 11000 * 0{,}215 = 28\ 380 \text{ грн} \quad (4.15)$$

Враховуючи додаткову заробітню плату:

$$C_{зп} = C_г * (1 + К_з) = 28\ 380 * (1 + 0{,}215) = 34\ 481 \text{ грн} \quad (4.16)$$

Далі відрахування на соціальний внесок:

$$C_{від} = C_{зп} * 0{,}22 = 7585{,}974 \text{ грн} \quad (4.17)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10 000 грн

$$C_A = 1{,}15 * 0{,}25 * 10000 = 2875 \text{ грн} \quad (4.18)$$

Підрахуємо витрати на профілактику та ремонт:

$$С_Р = К_{ТМ} * Ц_{ПР} * К_Р = 1,15 * 10000 * 0,05 = 575 \text{ грн} \quad (4.19)$$

Підрахуємо ефективний годинний фонд часу ПК за рік по формулі:

$$Т_{ЕФ} = (365 - 142 - 16) * 9 * 0,8 = 1490,4 \text{ год} \quad (4.20)$$

Розрахуємо витрати на електроенергію

$$С_{ЕЛ} = 1490,4 * 0,26 * 1,75 * 0,954 = 646,94 \text{ грн} \quad (4.21)$$

Накладні витрати рівні:

$$С_Н = 10000 * 0,67 = 6700 \text{ грн.} \quad (4.22)$$

Отже експлуатаційні витрати(грн):

$$С_{ЕКС} = 34\,481 + 7585,97 + 2875 + 575 + 646,94 + 6700 = 52\,863,9 \text{ грн} \quad (4.23)$$

Тоді собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{52\,863,9}{1490,4} = 35,47 \text{ грн/год} \quad (4.24)$$

Витрати на машину для варіантів:

$$С_м = 35,47 * 1174,432 = 41\,657,103 \quad (4.25)$$

$$С_м = 35,47 * 1564,344 = 55\,487,28 \quad (4.26)$$

Накладні витрати для варіантів:

$$C_Н = 102\,440 * 0{,}67 = 68634{,}8 \quad (4.27)$$

$$C_Н = 136\,567 * 0{,}67 = 91499{,}89 \quad (4.28)$$

Порахуємо повну вартість розробки для наших варіантів:

$$C_{ПП} = 102\,440 + 22536 + 41\,657{,}103 + 68634{,}8 = 235267{,}903 \quad (4.29)$$

$$C_{ПП} = 136\,567 + 30044 + 56677{,}363 + 91499{,}89 = 313\,598{,}17 \quad (4.30)$$

Знайдемо коефіцієнт техніко-економічного рівня використавши формулу:

$$К_{ТЕРj} = К_{Кj} / С_{Фj} \quad (4.31)$$

$$К_{ТЕР1} = 8{,}6726 / 235267{,}903 = 0{,}00003686, \quad (4.32)$$

$$К_{ТЕР2} = 7{,}9306 / 313\,598{,}17 = 0{,}00002529. \quad (4.33)$$

4.5 Висновки

За проведеними розрухунками робимо висновок що варіант з коефіцієнтом техніко-економічного рівня 0,00003686 є найбільш ефективним, це 1 варіант. Таким чином після проведного відбору(функціонально-вартісного аналізу) та підрахунків ми приймаємо рішення реалізовувати варіант один, котрий включає в себе використання навчальної вибірки обсягом 200000 текстів, CNN мережі та мови програмування Python 3. Так ми отримали оптимальний варіант дослідження та використання CNN у задачі роспізнавання тональності тексту.

ВИСНОВКИ

У даній дипломній роботі було виконано дослідження роботи штучної нейронної мережі CNN у задачі розпізнавання емоційного забарвлення тексту на прикладі коментарів у соціальній мережі Twitter, а також порівняння результатів її роботи з базовим рішенням у цій сфері – наївним баєсівським класифікатором.

Було виконано огляд теорії штучних нейронних мереж, починаючи з елементарних елементів та архітектур та закінчуючи складними алгоритмами та методами оптимізації їх роботи, а також різними архітектурами, у тому числі CNN.

Було побудовано CNN мережу та проведено випробування на підготованій для цього вибірці, а також задіяно певні прийоми для представлення тексту у векторному вигляді. Було розглянуто роботу моделі Word2Vec та використано цю заздалегідь навчену нами модель для роботи CNN.

На тій самій вибірці було навчено наївний баєсівський класифікатор, а також перевірено його роботу на тому самому наборі тестових даних.

Як результат конволюційна нейронна мережа показала кращий результат у порівнянні з басовим рішенням для такої задачі, а також було визначено шляхи її вдосконалення, та можливі корективи котрі можна внести у побудовану мережу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. A Survey of the Recent Architectures of Deep Convolutional Neural Networks URL :(https://arxiv.org/pdf/1901.06032.pdf) (дата звернення: 15.03.2020)
2. Штучні нейронні мережі :обчислення URL :(http://www.immsp.kiev.ua/postgraduate/Biblioteka_trudy/ShtuchnNejronMeregNester2004.pdf) (дата звернення: 20.04.2020)
3. A Comprehensive Guide to Convolutional Neural Networks URL :(https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53) (дата звернення: 10.04.2020)
4. Zhang Y., Wallace B. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification URL (https://arxiv.org/pdf/1510.03820.pdf) (дата звернення: 10.04.2020)
5. Шитиков В. К., Мастицкий С. Э. (2017) Классификация, регрессия, алгоритмы Data Mining с использованием R URL (https://ranalytics.github.io/data-mining/072-NBC.html) (дата звернення: 20.04.2020)

ДОДАТОК А

Лістинг програми

```
import pandas as pd
import numpy as np
n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw', 'faw', 'stcount', 'foll', 'frien',
'listcount']
data_positive = pd.read_csv('data/positive.csv', sep=';', error_bad_lines=False,
names=n, usecols=['text'])
data_negative = pd.read_csv('data/negative.csv', sep=';', error_bad_lines=False,
names=n, usecols=['text'])
sample_size = min(data_positive.shape[0], data_negative.shape[0])
raw_data = np.concatenate((data_positive['text'].values[:sample_size],
                    data_negative['text'].values[:sample_size]), axis=0)
labels = [1] * sample_size + [0] * sample_size
import re
def preprocess_text(text):
    text = text.lower().replace("ё", "е")
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', text)
    text = re.sub('@[^\s]+', 'USER', text)
    text = re.sub('[^a-zA-Za-яА-Я1-9]+', ' ', text)
    text = re.sub(' +', ' ', text)
    return text.strip()
data = [preprocess_text(t) for t in raw_data]
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=2)
import sqlite3
# Відкриваємо бд
```

```
conn = sqlite3.connect('mysqlite3.db')
c = conn.cursor()
with open('data/tweets.txt', 'w', encoding='utf-8') as f:
    # Читаємо текст
    for row in c.execute('SELECT ttext FROM sentiment'):
        if row[0]:
            tweet = preprocess_text(row[0])
            # Пишемо текст у файл
            print(tweet, file=f)
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
SENTENCE_LENGTH = 26
NUM = 100000
def get_sequences(tokenizer, x):
    sequences = tokenizer.texts_to_sequences(x)
    return pad_sequences(sequences, maxlen=SENTENCE_LENGTH)
tokenizer = Tokenizer(num_words=NUM)
tokenizer.fit_on_texts(x_train)
x_train_seq = get_sequences(tokenizer, x_train)
x_test_seq = get_sequences(tokenizer, x_test)
import logging
import multiprocessing
import gensim
from gensim.models import Word2Vec
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.INFO)
# Считуємо нерозмічені дані
data = gensim.models.word2vec.LineSentence('data/tweets.txt')
# Починаємо навчання Word2Vec
```

```
model = Word2Vec(data, size=200, window=5, min_count=3,
workers=multiprocessing.cpu_count())
model.save("models/w2v/model.w2v")
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
# Обмежуємо кількість слів у речені
SENTENCE_LENGTH = 26
NUM = 100000
def get_sequences(tokenizer, x):
    sequences = tokenizer.texts_to_sequences(x)
    return pad_sequences(sequences, maxlen=SENTENCE_LENGTH)
tokenizer = Tokenizer(num_words=NUM)
tokenizer.fit_on_texts(x_train)
x_train_seq = get_sequences(tokenizer, x_train)
x_test_seq = get_sequences(tokenizer, x_test)
from gensim.models import Word2Vec
# Завантажуємо модель
w2v_model = Word2Vec.load('models/w2v/model.w2v')
DIM = w2v_model.vector_size
embedding_matrix = np.zeros((NUM, DIM))
# Додаємо NUM слів
for word, i in tokenizer.word_index.items():
    if i >= NUM:
        break
    if word in w2v_model.wv.vocab.keys():
        embedding_matrix[i] = w2v_model.wv[word]
from keras.layers import Input
from keras.layers.embeddings import Embedding
tweet_input = Input(shape=(SENTENCE_LENGTH,), dtype='int32')
tweet_encoder = Embedding(NUM, DIM, input_length=SENTENCE_LENGTH,
```

```
                    weights=[embedding_matrix], trainable=False)(tweet_input)
from keras import backend as K
def precision(y_true, y_pred):
    """Precision metric.
    Only computes a batch-wise average of precision.
    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
def recall(y_true, y_pred):
   """Recall metric.
    Only computes a batch-wise average of recall.
    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall



def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        """Recall metric.
        Only computes a batch-wise average of recall.
        Computes the recall, a metric for multi-label classification of
        how many relevant items are selected.
```

```
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall
    def precision(y_true, y_pred):
        """Precision metric.
        Only computes a batch-wise average of precision.
        Computes the precision, a metric for multi-label classification of
        how many selected items are relevant.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2 * ((precision * recall) / (precision + recall + K.epsilon()))
from keras import optimizers
from keras.layers import Dense, concatenate, Activation, Dropout
from keras.models import Model
from keras.layers.convolutional import Conv1D
from keras.layers.pooling import GlobalMaxPooling1D
branches = []
# Добавляем dropout-регуляризацию
x = Dropout(0.2)(tweet_encoder)
for size, filters_count in [(2, 10), (3, 10), (4, 10), (5, 10)]:
    for i in range(filters_count):
        # Додаємо згортку
```

```
        branch = Conv1D(filters=1, kernel_size=size, padding='valid',
activation='relu')(x)
        # Додаємо субдекскритизацію
        branch = GlobalMaxPooling1D()(branch)
        branches.append(branch)
# Поєднуємо ознаки
x = concatenate(branches, axis=1)
# Додаємо дропаут
x = Dropout(0.2)(x)
x = Dense(30, activation='relu')(x)
x = Dense(1)(x)
output = Activation('sigmoid')(x)
model = Model(inputs=[tweet_input], outputs=[output])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[precision,
recall, f1])
model.summary()
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model.png')
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint("models/cnn/cnn-frozen-embeddings-
{epoch:02d}-{val_f1:.2f}.hdf5",
                             monitor='val_f1', save_best_only=True, mode='max', period=1)
history = model.fit(x_train_seq, y_train, batch_size=32, epochs=10,
validation_split=0.25, callbacks = [checkpoint])
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
def plot_metrix(ax, x1, x2, title):
    ax.plot(range(1, len(x1) + 1), x1, label='train')
    ax.plot(range(1, len(x2) + 1), x2, label='val')
```

```
    ax.set_ylabel(title)
    ax.set_xlabel('Epoch')
    ax.legend()
    ax.margins(0)
def plot_history(history):
    fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(16, 9))
    ax1, ax2, ax3, ax4 = axes.ravel()
    plot_metrix(ax1, history.history['precision'], history.history['val_precision'],
'Precision')
    plot_metrix(ax2, history.history['recall'], history.history['val_recall'], 'Recall')
    plot_metrix(ax3, history.history['f1'], history.history['val_f1'], "$F_1$")
    plot_metrix(ax4, history.history['loss'], history.history['val_loss'], 'Loss')
    plt.show()
plot_history(history)
model.load_weights('models/cnn/cnn-frozen-embeddings-09-0.76.hdf5')
from sklearn.metrics import classification_report
predicted = np.round(model.predict(x_test_seq))
print(classification_report(y_test, predicted, digits=5))
from keras import optimizers
model.layers[1].trainable = True
adam = optimizers.Adam(lr=0.0001)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=[precision,
recall, f1])
model.summary()
checkpoint = ModelCheckpoint("models/cnn/cnn-trainable-{epoch:02d}-
{val_f1:.2f}.hdf5",monitor='val_f1', save_best_only=True, mode='max', period=1)
history_trainable = model.fit(x_train_seq, y_train, batch_size=32, epochs=5,
validation_split=0.25, callbacks = [checkpoint])
model.load_weights('models/cnn/cnn-trainable-05-0.77.hdf5')
predicted = np.round(model.predict(x_test_seq))
```

```
print(classification_report(y_test, predicted, digits=5))
print(predicted)
print(x_test_seq)
plot_history(history_trainable)
n = ['id', 'date','name','text','typr','rep','rtw','faw','stcount','foll','frien','listcount']
data_positive = pd.read_csv('data/positive.csv', sep=';',error_bad_lines=False,
names=n, usecols=['text'])
data_negative = pd.read_csv('data/negative.csv', sep=';',error_bad_lines=False,
names=n, usecols=['text'])
sample_size = min(data_positive.shape[0], data_negative.shape[0])
raw_data = np.concatenate((data_positive['text'].values[:sample_size],
                           data_negative['text'].values[:sample_size]), axis=0)
labels = [1]*sample_size + [0]*sample_size
import re

def preprocess_text(text):
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL', text)
    text = re.sub('@[^\s]+','USER', text)
    text = text.lower().replace("ё", "е")
    text = re.sub('[^a-zA-Za-яА-Я1-9]+', ' ', text)
    text = re.sub(' +',' ', text)
    return text.strip()

data = [preprocess_text(t) for t in raw_data]
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.model_selection import train_test_split, GridSearchCV

text_clf = Pipeline([('vect', CountVectorizer()),
```

```
                ('tfidf', TfidfTransformer()),
                ('clf', MultinomialNB())])

tuned_parameters = {
   'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
   'tfidf__use_idf': (True, False),
   'tfidf__norm': ('l1', 'l2'),
   'clf__alpha': [1, 1e-1, 1e-2]
}
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.33,
random_state=42)
from sklearn.metrics import classification_report

score = 'f1_macro'
print("# Tuning hyper-parameters for %s" % score)
print()
np.errstate(divide='ignore')
clf = GridSearchCV(text_clf, tuned_parameters, cv=10, scoring=score)
clf.fit(x_train, y_train)

print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Grid scores on development set:")
print()
for mean, std, params in zip(clf.cv_results_['mean_test_score'],
                             clf.cv_results_['std_test_score'],
                             clf.cv_results_['params']):
   print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
print()

print("Detailed classification report:")
print()
print("The model is trained on the full development set.")
print("The scores are computed on the full evaluation set.")
print()
print(classification_report(y_test, clf.predict(x_test), digits=4))
print()
```

ДОДАТОК Б
Ілюстративний матеріал

Рисунок 1 - Вступний слайд

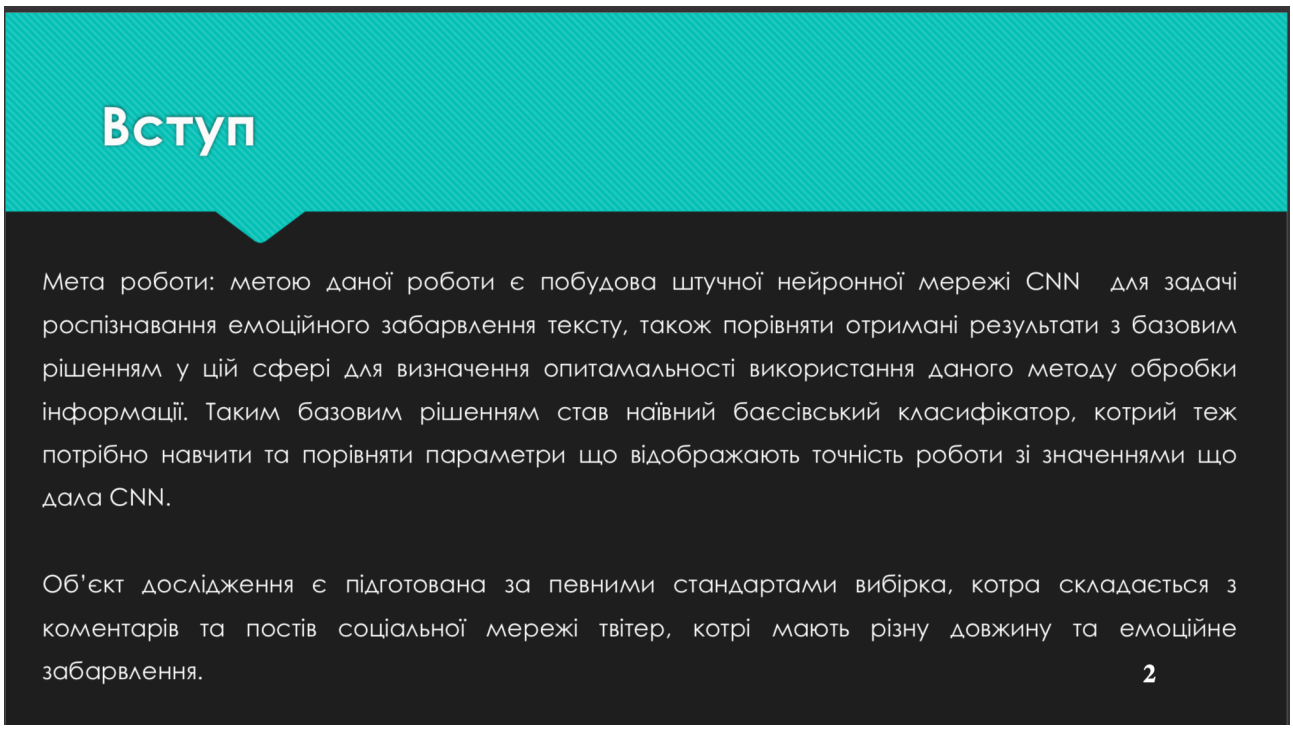


Рисунок 2 - Мета роботи

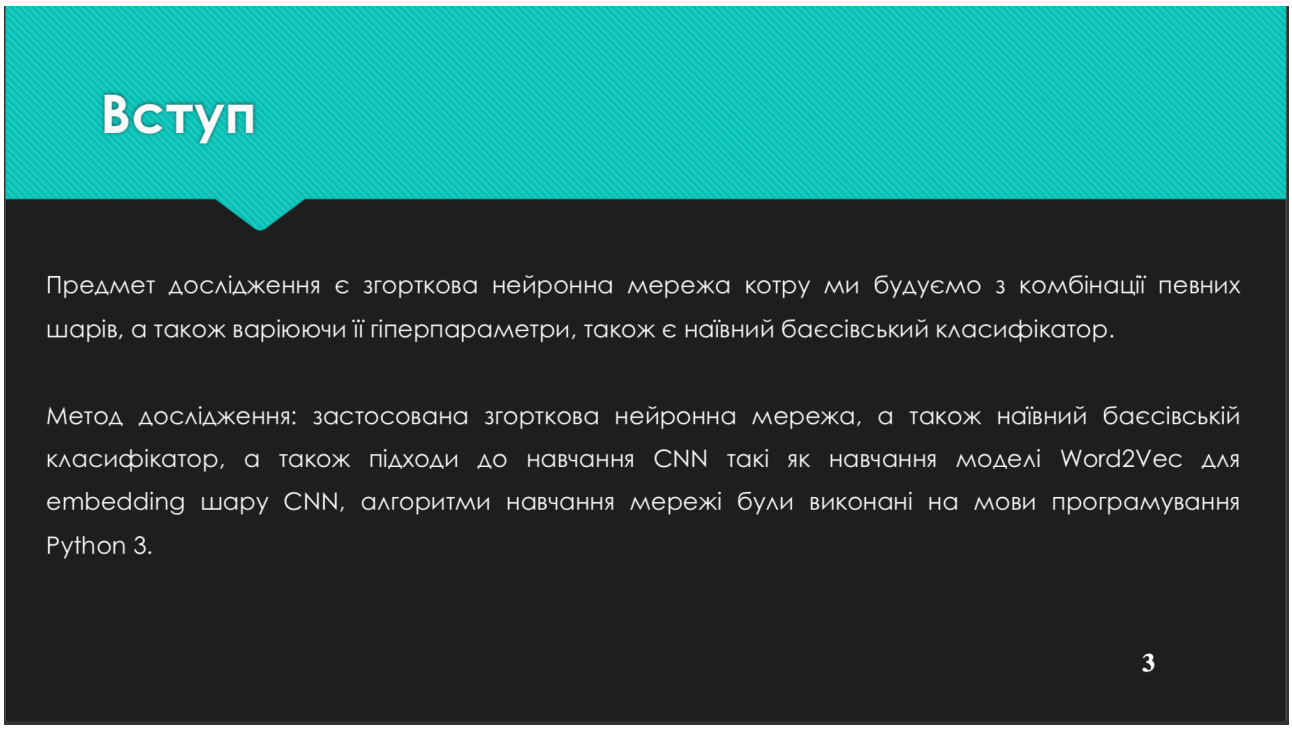


Рисунок 3 - Мета роботи

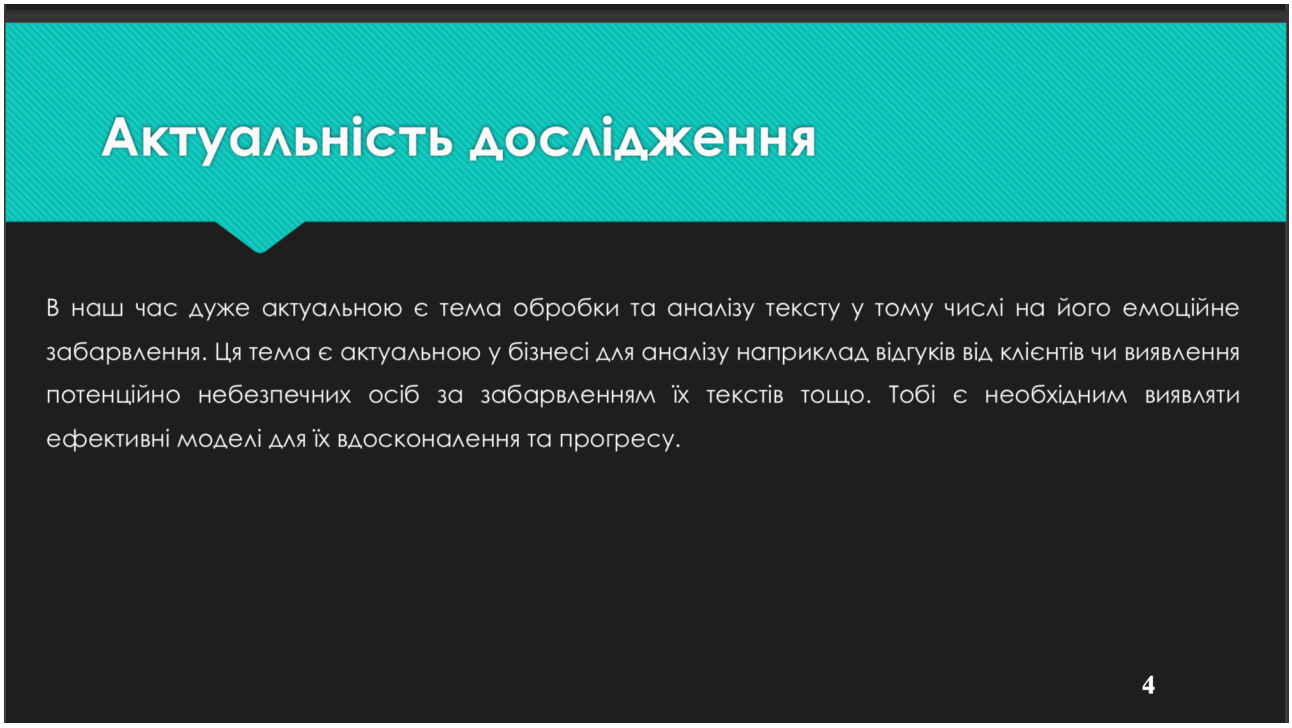


Рисунок 4 - Актуальність

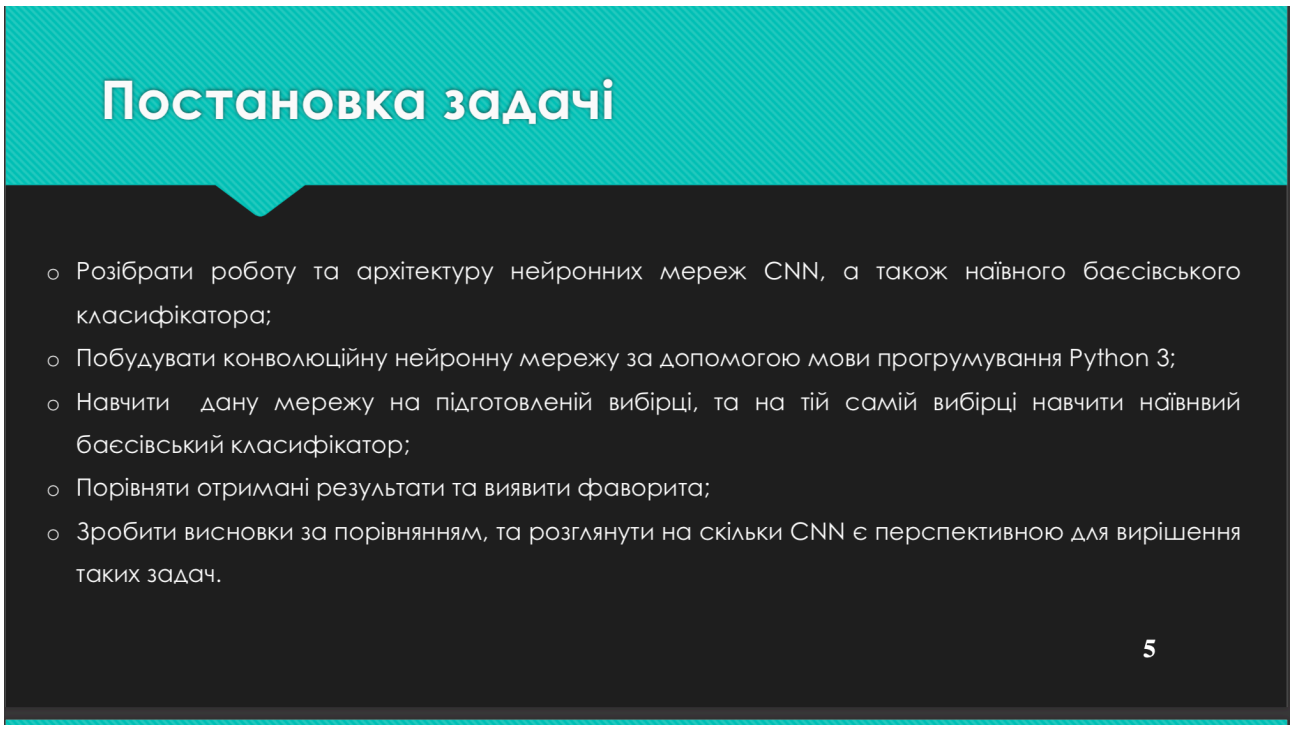


Рисунок 5 - Постановка задачі

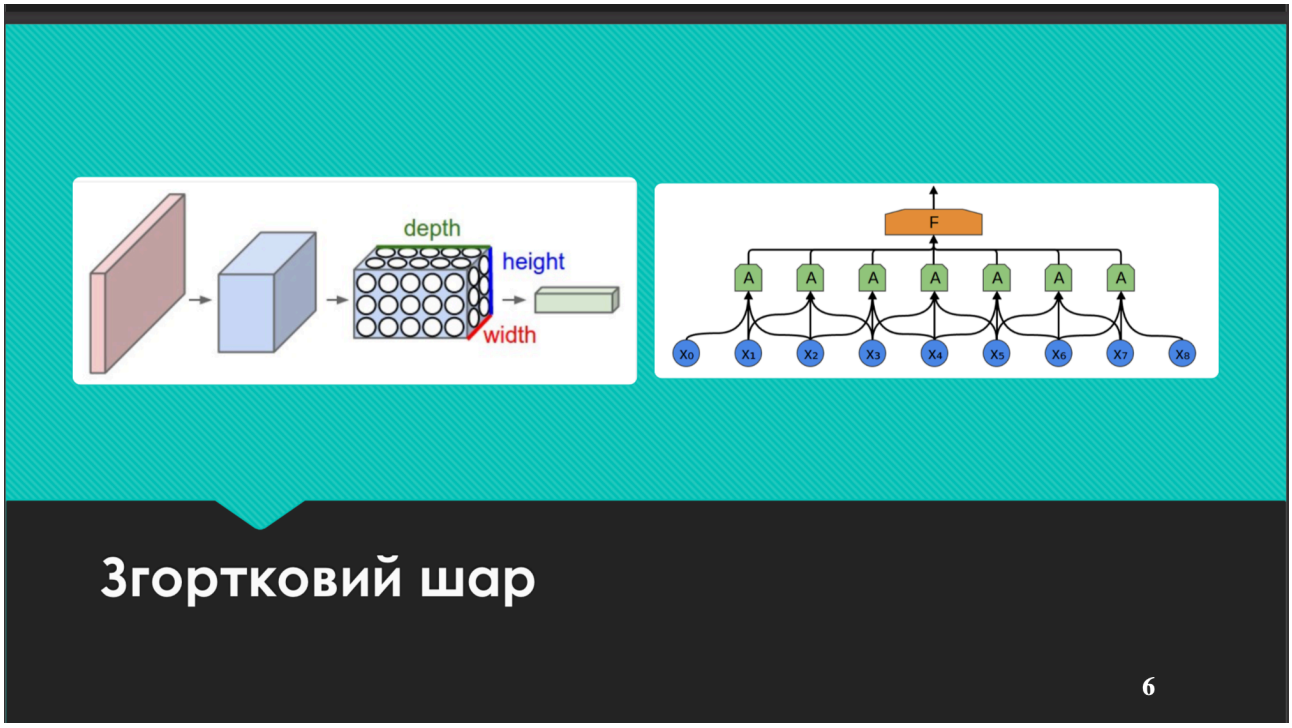


Рисунок 6 - Згортковий шар

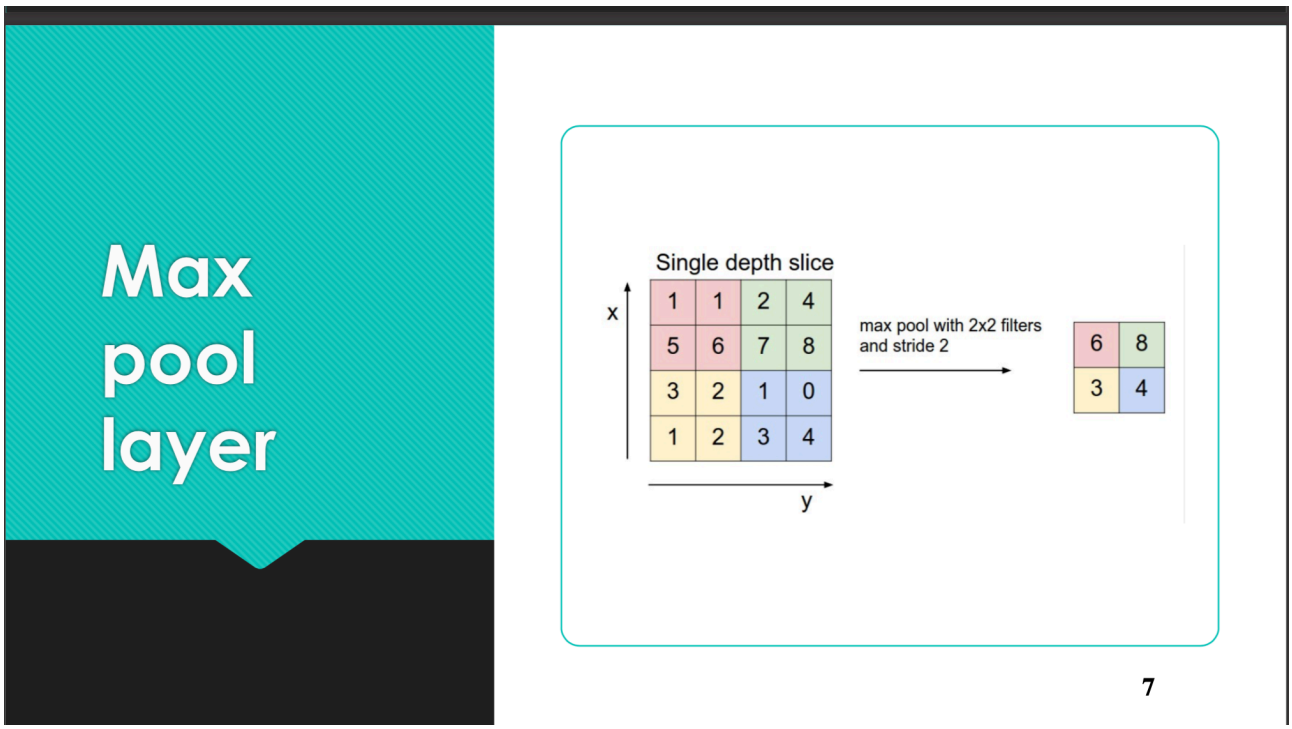


Рисунок 7 – Max-pooling

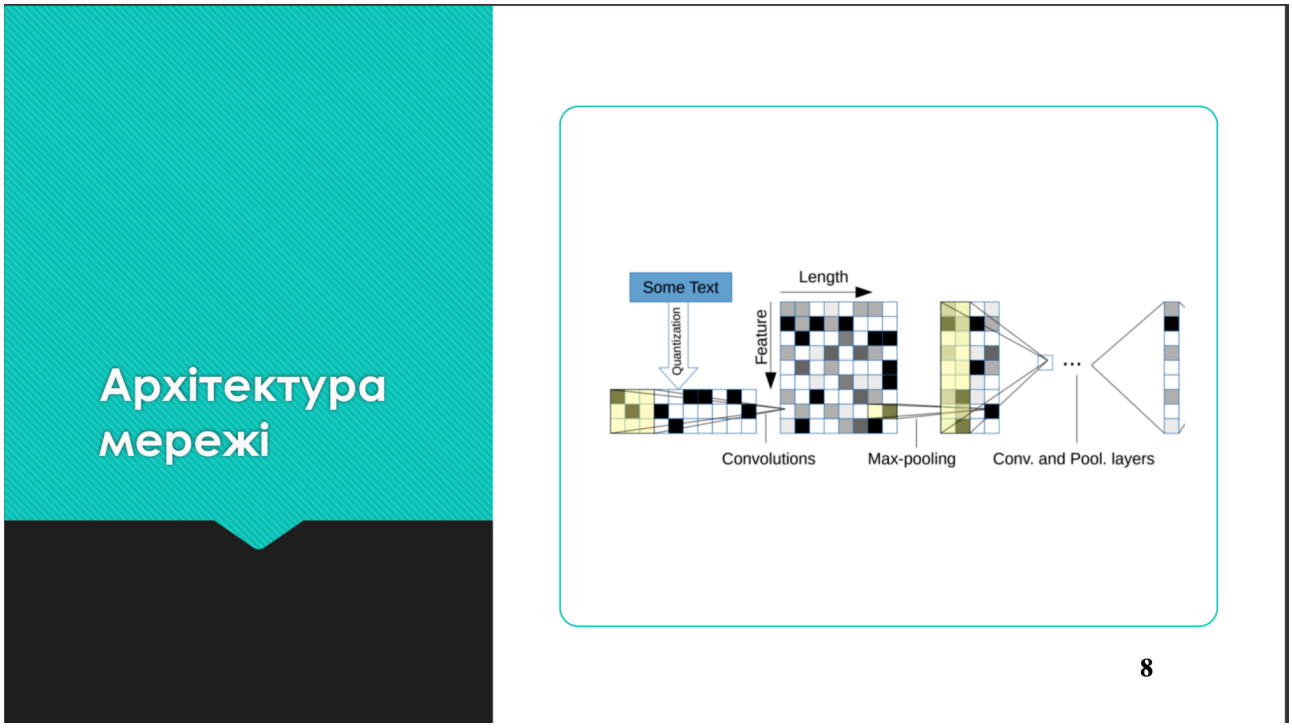


Рисунок 8 – Архітектура мережі

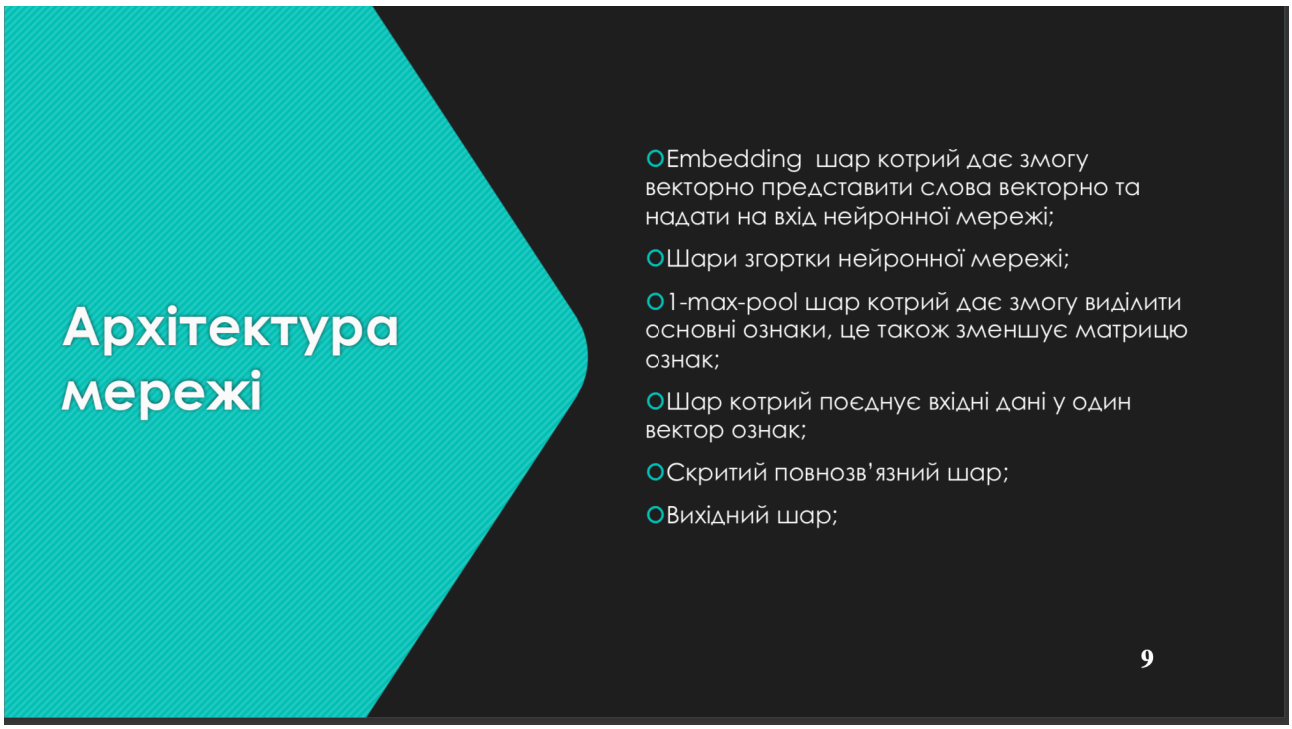


Рисунок 9 – Архітектура мережі (2)

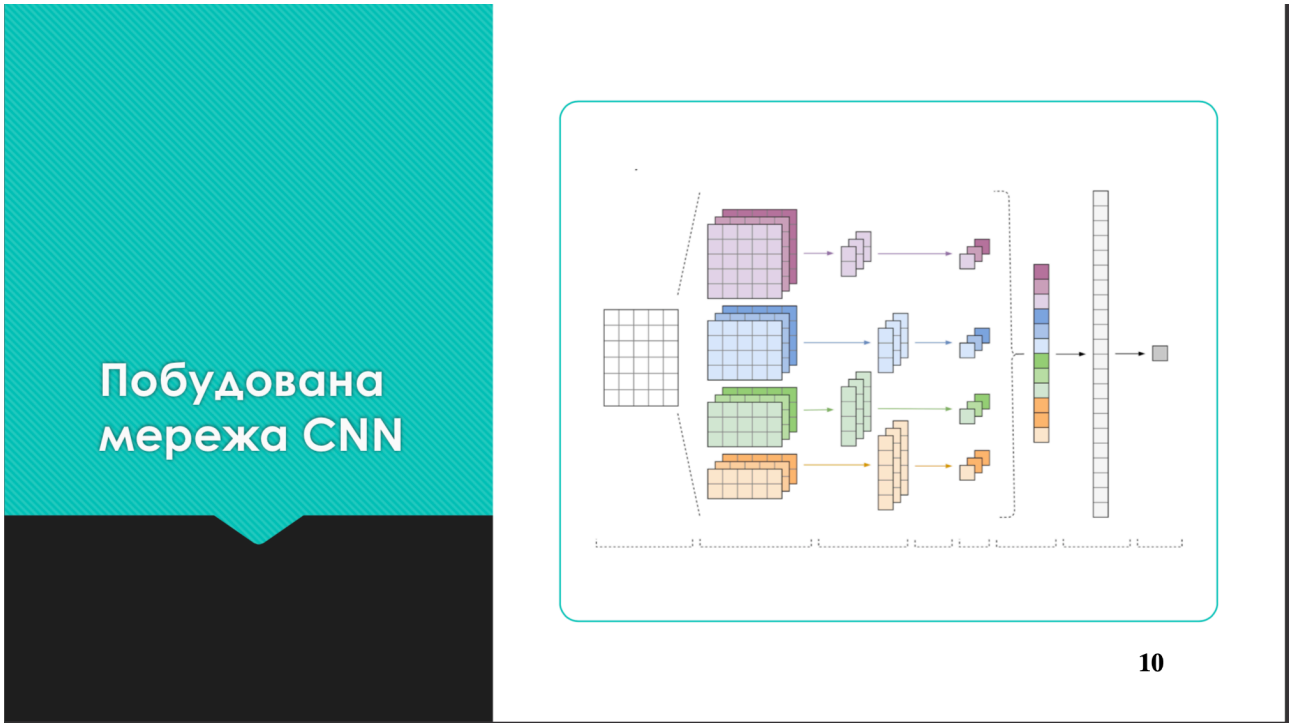


Рисунок 10 – Побудована мережа CNN

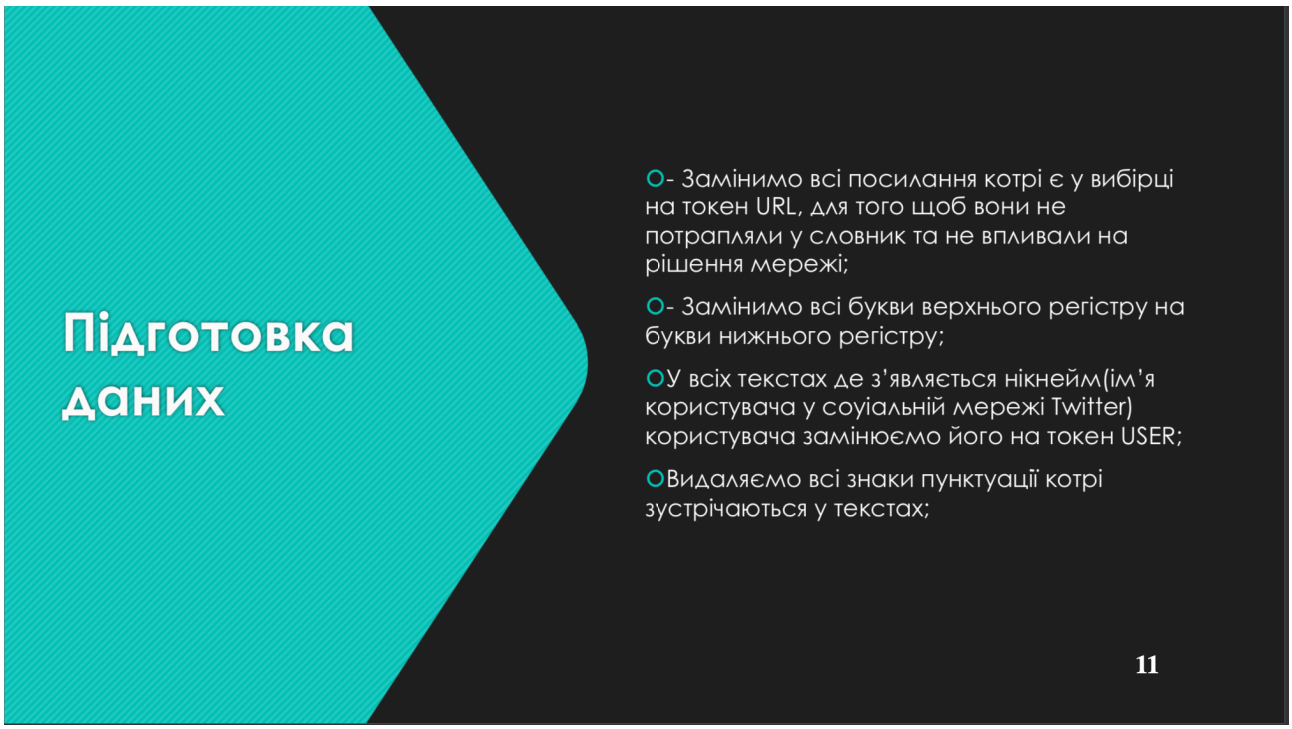


Рисунок 11 – Підготовка даних

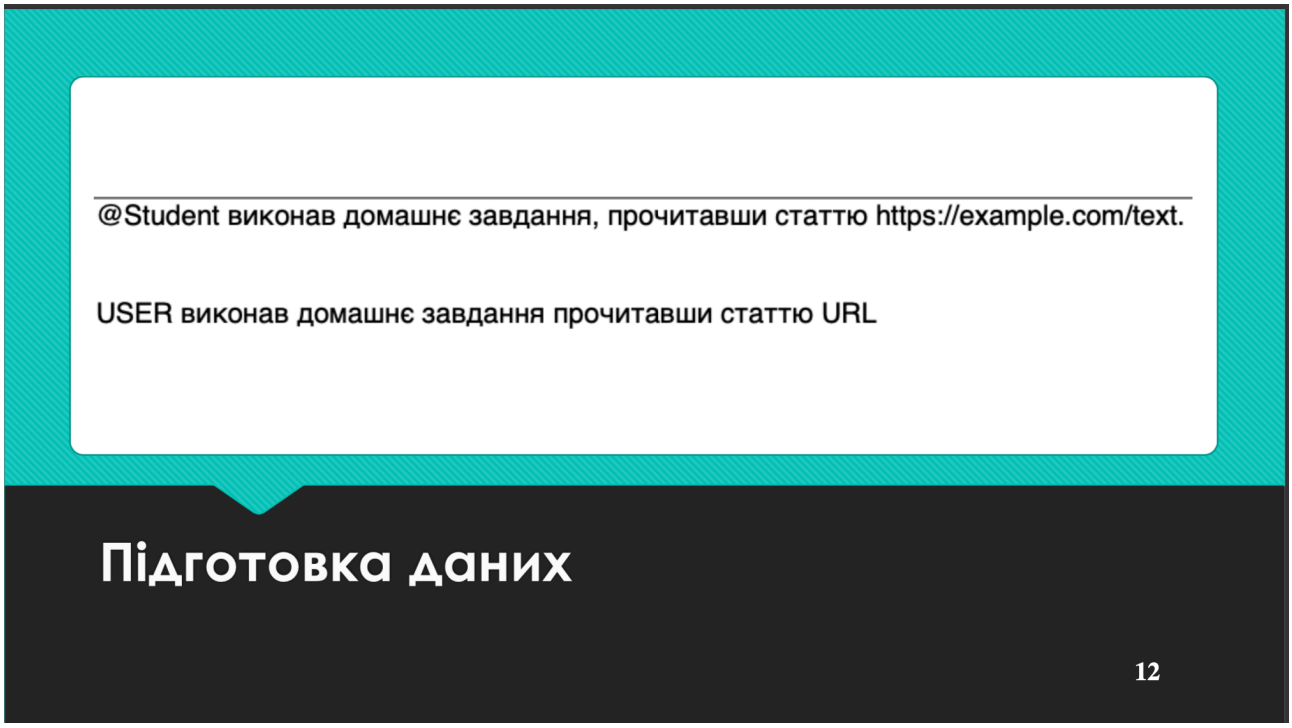


Рисунок 12 – Підготовка даних (2)

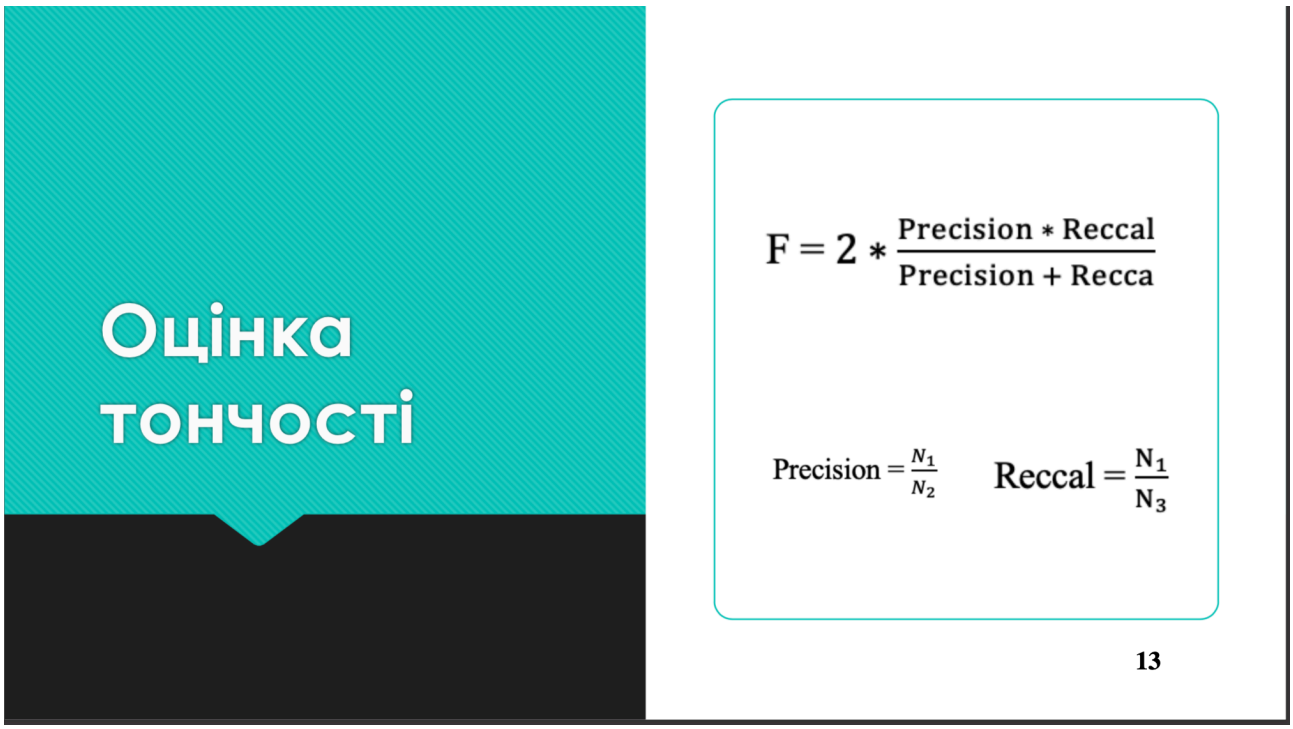


Рисунок 13 – Оцінка точності

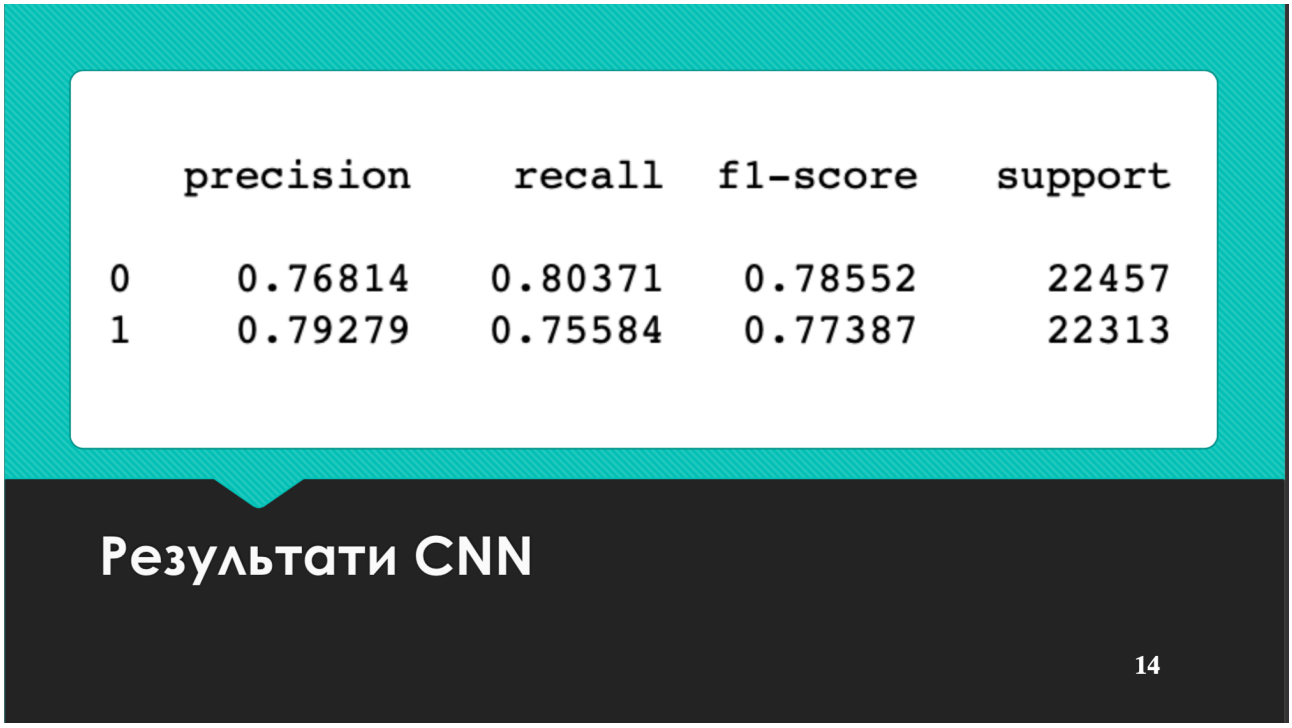


Рисунок 14 – Результати CNN

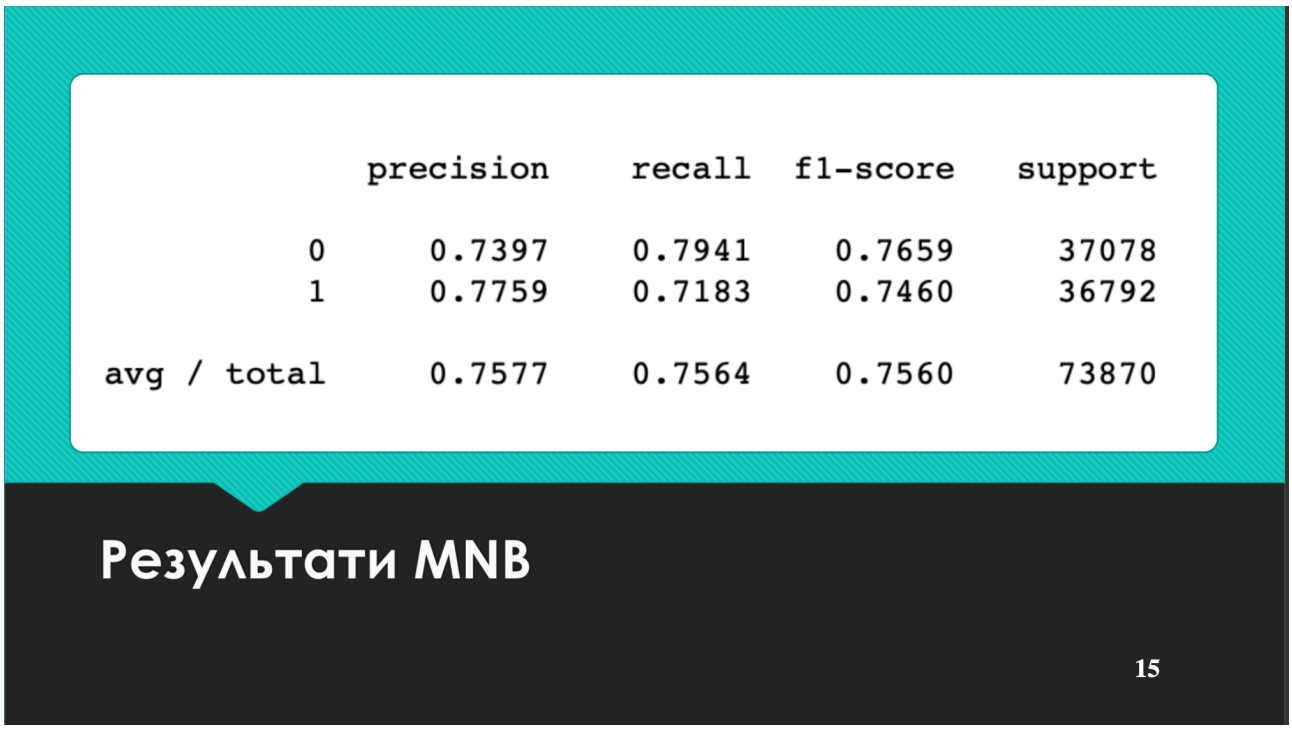


Рисунок 15 – Результати MNB

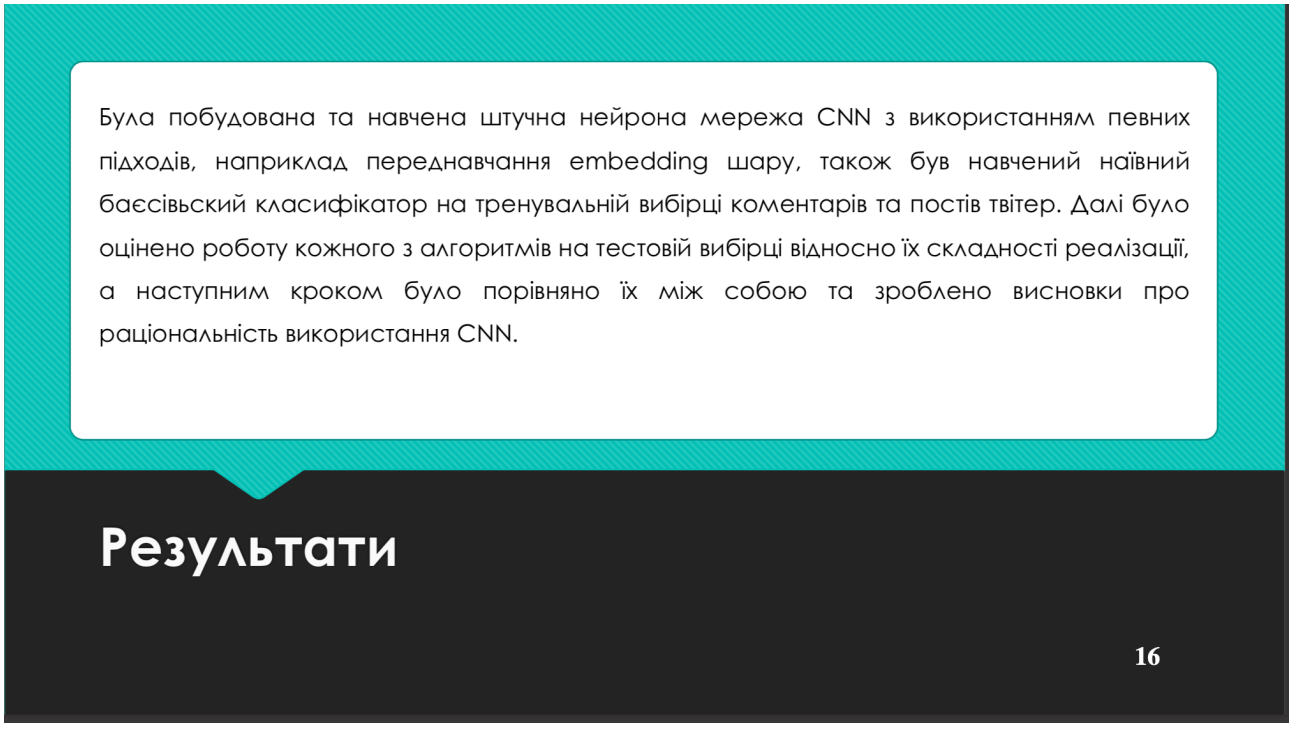


Рисунок 16 – Результати

Класифікатор	Precision	Recall	F1
CNN	0,7804	0,7798	0,7796
MNB	0,7577	0,7564	0,756

Порівняння результатів

17

Рисунок 17 – Порівняння результатів

Дякую за увагу!

18

Рисунок 18 – Завершальний слайд